Master of Science Thesis
Department of Computer Science
Lund Institute of Technology

Mui: Controlling Equipment via Migrating User Interfaces

Torbjörn Eklund <d97te@efd.lth.se>
David Svensson <d98dsv@efd.lth.se>

December 2002

Master of Science in Computer Science Engineering

Supervisors: Görel Hedin, Lund Institute of Technology
Boris Magnusson, Lund Institute of Technology
Mats Iderup, BlueCell AB, Lund
Björn Strandmark, BlueCell AB, Lund

The names of companies and products in this document may be the trademarks of their respective owners.

Abstract

This master thesis is about controlling equipment via migrating user interfaces. A handheld device, such as a Bluetooth-equipped cellular phone or handheld computer, is used as a generic "remote control". When the user gets in the vicinity of controllable equipment, the equipment is automatically discovered and a user interface is downloaded, if the user so wishes. This enables the user to interact with the equipment.

Possible fields of application are, e.g., controlling home electronics or heating systems. We have built a prototype where a video cassette recorder is controlled from a Palm handheld computer.

Much effort was spent making the user interface description flexible, so that the user interfaces could be displayed on devices which differ in screen sizes.

Acknowledgements

We would like to thank our supervisors at BlueCell, Mats Iderup and Björn Strandmark. They have devoted a lot of time to us and they have always been very patient with our questions.

Our supervisors at Lund Institute of Technology, Görel Hedin and Boris Magnusson, have been very helpful and have helped us to stay focused on what was important with the project so we did not get lost in small details.

We would like to thank ConnectBlue who donated the Wireless Information Provider for use in this master thesis.

We would like to express our gratitude to Anders Ive who answered our questions about Java Virtual Machines. He also made the short film which we showed at our presentation. This short film can be downloaded from `http://www.lucas.lth.se/new/research/demos/mui/index.shtml`.

Thanks go to Joakim Roubert who provided us with the template LaTeX document which we have used when writing the report. He also starred in the short film.

We would like to thank the proofreaders Barbro Ericsson, Jacob Gradén, Jens Gulin, Markus Helgesson, Eric Michell, Markus Olausson, and Joakim Roubert.

Finally, we would also like to thank Magnus Bäck, Fredrik Löfgren, and Jacob Nordgren.

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 The Vision . . . 2
- 1.3 Our Work . . . 3
- 1.4 Related Work . . . 5
- 1.5 Roadmap . . . 5

2 Architecture **7**
- 2.1 Use-Cases . . . 7
- 2.2 Components in the System . . . 8
- 2.3 Flow of Events . . . 9

3 Java on Small Devices **11**
- 3.1 Why Java? . . . 11
- 3.2 J2ME and MIDlets . . . 12
- 3.3 Choosing a Virtual Machine . . . 13

4 User Interfaces for General Handheld Devices **17**
- 4.1 Java GUIs on Desktop Computers . . . 19
- 4.2 Existing Solutions . . . 20
  - 4.2.1 MIDP . . . 20
  - 4.2.2 kAWT . . . 20
  - 4.2.3 UIML . . . 21
- 4.3 Our Library: `muigui` . . . 23

5 Implementation **27**
- 5.1 Description of the Hardware . . . 28
  - 5.1.1 Palm Handheld . . . 28
  - 5.1.2 Palm Bluetooth Card . . . 29
  - 5.1.3 Wireless Information Provider . . . 29
  - 5.1.4 Programmable Logic Controller . . . 30
  - 5.1.5 Video Cassette Recorder . . . 30
- 5.2 Components in the System . . . 32
- 5.3 Flow of Events . . . 33
- 5.4 MuiDiscovery . . . 34

5.4.1 Discovering Services . . . 35
5.4.2 Downloading and Installation . . . 36
5.4.3 Starting the MIDlet . . . 37
5.5 Server Device . . . 38
5.6 Remote Control MIDlet . . . 41

6 Evaluation **45**
6.1 Components in the System . . . 45
6.1.1 Client Device . . . 45
6.1.2 Server Device . . . 48
6.2 Security . . . 50
6.3 Usability . . . 50

7 Related Work **53**
7.1 Research Project . . . 53
7.2 JayTalk . . . 53
7.3 Jini . . . 55
7.4 UbiControl . . . 55
7.5 Wireless Microservers . . . 56
7.6 The Pebbles Project . . . 56

8 Conclusions and Further Work **59**
8.1 Conclusions . . . 59
8.2 Further Work . . . 60

A User Manuals **61**
A.1 MuiDiscovery . . . 61
A.2 SonyVideo . . . 63

B `muigui` Overview **65**

C Bluetooth **67**
C.1 Background . . . 67
C.2 Profiles . . . 67
C.3 The Technology . . . 68

D Dictionary **69**

Chapter 1

Introduction

1.1 Background

We are heading towards a society where more and more gadgets are equipped with computing power and wireless communication. An expression used in this context is *ubiquitous computing*, ubicomp. This term, coined by Mark Weiser in 1988 [38], means that computers are present everywhere in our daily environment. However, they should barely be noticeable for the user, who should be able to interact with them when wanted without being disturbed at other times. Wireless communication is important to achieve this goal. The term *pervasive computing* is also used, as a synonym to ubiquitous computing.

As a first example, we take a look at the home. There, the traditional way to control devices wirelessly is by a remote control. Standard remote controls use IR, while current technologies for wireless communication often use radio—an example of this is Bluetooth. The main difference between IR and radio in remote controls is that IR requires free line of sight to work. Furthermore, traditional IR remote controls often only give one-way communication, while radio works both ways.

When the number of controllable devices in houses increases, the same happens with the number of remote controls—the home will be overwhelmed with remote controls. A way out of this is to replace the remote controls with a single device: the user's own cellular phone or handheld computer. There are several advantages of this replacement, apart from the decrease in remote controls. In contrast to ordinary remote controls, cellular phones and handheld computers can present dynamic information, and adjust the user interface to the state of the controlled device. Two-way communication allows information to be sent from the controlled device to the remote control. E.g., the image shown on a TV could also be shown on the remote control.

A remote control with these properties is suitable also in new situations where remote controls are normally not used. One example could be

the heating system in a building; today it is controlled by a dedicated box located on a wall with buttons and a display. The two-way communication would make it possible for the heating system to display information to the user by sending a message to the remote control device when a property changes in the heating system and then the need for a box with buttons decreases.

To achieve the highest degree of flexibility, the cellular phone or handheld computer should have to be configured as little as possible in advance to be used as a remote control. The adaption should be at a general level and not be specific for a certain kind of controlled device.

This project is the first step in a research project at the department of Computer Science at Lund Institute of Technology. Our part is based on previous experience from BlueCell, the company where all the thesis work has been carried out. The ideas behind this project come from BlueCell and from earlier projects at the department of Computer Science.

1.2 The Vision

The vision behind this thesis is as follows. The user of the system carries his cellular phone or handheld computer, in the remainder of this text referred to as the *client device*. He approaches a controllable device, the *server device*, e.g. a TV. The client device informs the user that there is a server device nearby. The user might choose to download the user interface and can then begin to use the *service*, to control the server device.

There are several advantages of keeping the user interface on the server device and downloading it when needed. A first advantage is that the user interface is always available and installs itself. The correct version is always used and the user does not have to download it from the Internet or install it from a CD-ROM. A second advantage is that the manufacturer can customise the user interface to fully use all features of the particular server device. A third advantage is that each server can decide its own protocol for the communication between the client device and server device[1]. It is not necessary to follow a standardised remote control protocol. The protocol can be changed easily without backward compatibility problems.

To exemplify, we will give three scenarios, which will serve to give an idea of the concept.

- Bob has checked into a hotel. In his room, he looks at his cellular phone and discovers that he can access several services in the room. He chooses to control the TV and gets a remote control interface on

[1]The communication always uses the same base protocol, e.g. TCP/IP, but the server can decide the protocol on top of that, for sending commands and responses between the client device and server device.

the screen of the cellular phone. He switches between the channels and adjusts the volume.

- Alice is a technician, working with repairs and maintenance of ventilation systems. She has been called to a building where the air conditioning is malfunctioning. When Alice enters the building, she picks up her handheld computer and the ventilation system pops up as an icon on the handheld. Alice starts the service and gets a map of the ventilation system on the screen. She gets status information about different parts of the system and is able to zoom into the area that causes problems and finds the error. Alice shuts down the affected area from the handheld computer while fixing the problem.

- Chuck has borrowed a friend's digital camera for a vacation. He has taken a nice picture, which he wants to share with his friend. He turns on his cellular phone, which discovers the digital camera. Chuck starts an image service. With it, he is able to browse the pictures on the camera and select the one he is interested in. He edits the picture and attaches it to an e-mail, which he then sends to his friend.

1.3 Our Work

The project has been named Mui, which stands for Migrating User Interfaces. The goal was to investigate the possibilities to construct a system as discussed above and build a prototype for a simple scenario. We chose to control a Sony VCR (video cassette recorder) from a Palm handheld computer using Bluetooth, a radio technology for short-range communication. The handheld computer should be used as a remote control, as shown in Figure 1.1.

The user starts an application on the client device which, using Bluetooth, finds all nearby services that the system supports. The user can download and run a user interface if he finds a service he is interested in. Two screenshots of these situations can be found in Figure 1.2.

One partial goal when building the system was to make the user interface able to work on many different client devices. We used Java to achieve this, since Java is designed to be platform independent. One problem is that the screen sizes of the different client devices may differ a lot. They also vary in how they receive input from the user. These two problems make it difficult to construct a user interface that works well on many devices. This was approached in a user interface library, `muigui`, that we created.

Figure 1.1: *Palm handheld and Sony VCR.*

Figure 1.2: *The left figure shows a list of services on the Palm handheld. The user can choose one, and a user interface is downloaded and shown to the user as in the right figure.*

1.4 Related Work

As mentioned above, the project builds on previous projects at BlueCell. It is strongly connected to BlueCell's product JayTalk. Sun's Jini technology is also relevant in this context. These technologies will be discussed in Chapter 7. There, we will also look at UbiControl and Wireless Microservers, two related projects that use handheld devices as remote controls. The research project at the department of Computer Science at Lund Institute of Technology, for which our work is a preliminary study, will also be discussed.

1.5 Roadmap

The rest of this report is structured as follows:

Chapter 2, Architecture, gives a brief description of the overall architecture of our solution.

Chapter 3, Java on Small Devices, discusses how to use Java on devices with limited resources and our choice of Java Virtual Machine.

Chapter 4, User Interfaces for General Handheld Devices, is about problems faced when constructing user interfaces intended for many different devices.

Chapter 5, Implementation, describes the implementation of the prototype system.

Chapter 6, Evaluation, evaluates the implementation and discusses the choices that were made.

Chapter 7, Related Work, is a survey of some work in this area and related technologies.

Chapter 8, Conclusions and Further Work, contains a summary of our results and ideas for future work.

Chapter 2

Architecture

In this chapter an overview of the architecture of the solution will be given. It will be a generic description, not targeted for a special platform. The components needed in the different devices and how they should work together have been identified. Details about how the components were implemented can be found in Chapter 5.

2.1 Use-Cases

Two main use-cases have been identified and the architecture targets these use-cases. They are more specific than the scenarios given in Section 1.2.

Discovering and starting a service: The user holds the client device in the proximity of a server device. The client device discovers the server device and asks for a list of services, which is downloaded and displayed on the client device. The user selects a service from the list. A request is made by the client device for the graphical user interface, which is downloaded from the server device. The graphical user interface is then displayed on the client device.

Using a service: A service has been started on the client device. The user issues a command by making an action in the graphical user interface. The command is sent to the server device and executed. A response is sent back and is displayed to the user on the screen of the client device. This sequence can be repeated numerous times if the user wants to issue more commands.

2.2 Components in the System

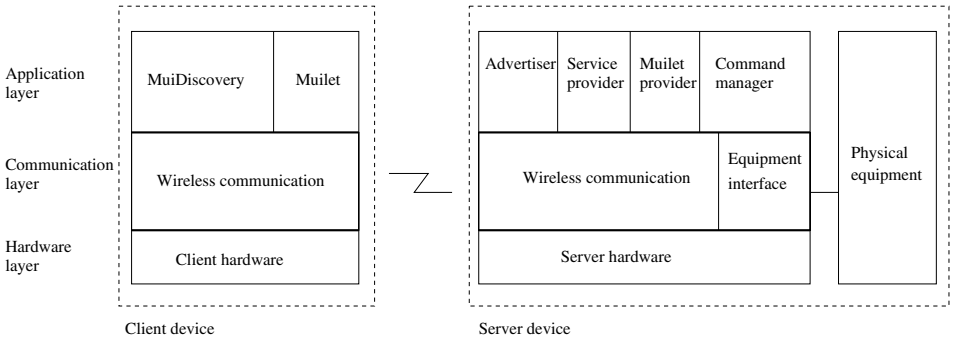

Figure 2.1: *A layered view of the architecture which shows the components in the server device and client device.*

Figure 2.1 presents a layered view of the architecture. Upper layers use functionality provided by lower layers. The layers are the following:

Hardware layer: Contains the actual hardware that the upper layers are running on.

Communication layer: Responsible for the communication with other devices.

Application layer: Software components that handle Mui services. The user interacts with the application layer on the client device, using the input and output mechanisms of the client device.

To implement the use-cases in Section 2.1, a number of components must be present in the client device and server device. The components in the client device are the following:

Client hardware: The hardware of the client device, e.g. a handheld computer or cellular phone.

Wireless communication: Responsible for the wireless communication with the server device.

MuiDiscovery: Finds services in the vicinity of the client device and lets the user start them.

Muilet: A small application, that shows the graphical user interface of a service on the client device and contains logic for controlling the server device. This component is downloaded from the server device when needed.

The server device represents the equipment the user can control (the physical equipment) and additional components needed to make the user interface available to client devices. These additional components are:

Server hardware: The hardware needed to make the user interface available to client devices. The software components of upper layers run on this hardware.

Wireless communication: Responsible for the wireless communication with the client device.

Equipment interface: Interface to the physical equipment.

Advertiser: Advertises the server device to client devices, saying that the device offers Mui services available for download.

Service provider: Gives a list of services to the client device, when asked.

Muilet provider: Provides the client device with the Muilet for a service.

Command manager: Receives commands from the client device and executes them on the physical equipment through the equipment interface.

2.3 Flow of Events

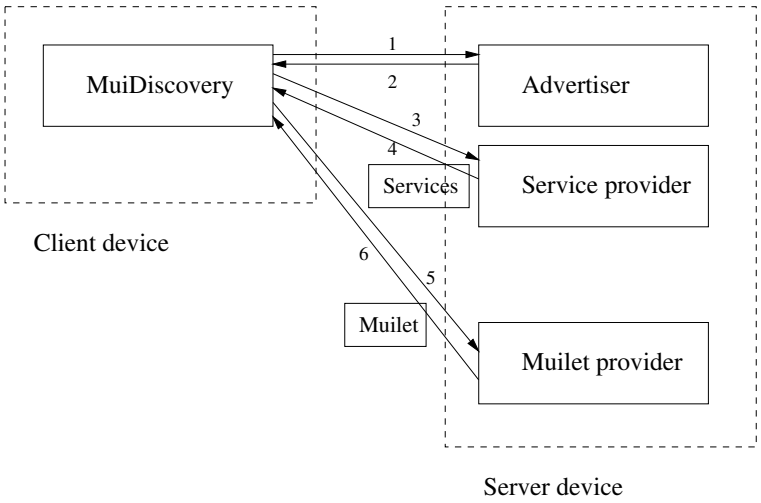

Figure 2.2: *The flow of events for the use-case "Discovering and starting a service" in Section 2.1. An explanation of the numbers is found in Section 2.3.*

The flow of events for the use-case "Discovering and starting a service" in Section 2.1 is illustrated in Figure 2.2. The steps are the following:

1. MuiDiscovery searches for server devices in the vicinity. One server device is found.
2. The Advertiser in the found server device responds and a wireless connection is established.
3. MuiDiscovery asks for a list of services.
4. Service provider returns the list to MuiDiscovery. The list is shown to the user.
5. The user selects one service and MuiDiscovery asks the Muilet provider for that service's Muilet.
6. The Muilet is transferred to MuiDiscovery, which starts it. The GUI is shown to the user.

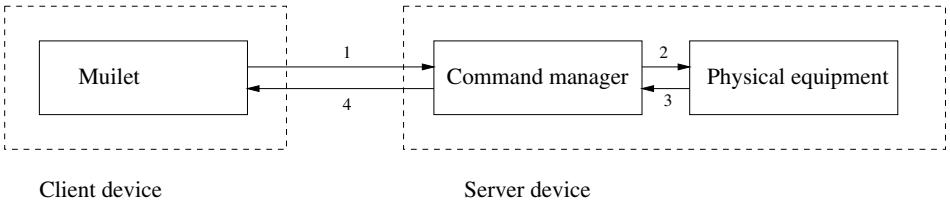

Figure 2.3: *The flow of events for the use-case "Using a service" in Section 2.1. An explanation of the numbers is found in Section 2.3.*

The use-case "Using a service" continues where the first use-case ends. It has the following flow of events, as shown in Figure 2.3:

1. The user chooses a command in the GUI and the command is sent to the Command manager.
2. The Command manager executes the command on the physical equipment.
3. The physical equipment responds with status information about the result of the command.
4. The status information is sent back to the client device and shown to the user.

Chapter 3

Java on Small Devices

We have chosen to implement the GUI in Java. The Muilet component, described in Section 2.2, is a Java application. For the Muilet to execute on the client device a Java Virtual Machine (JVM) and a graphical library are needed. A JVM is a software component that acts as a software processor and executes Java programs. This chapter discusses running Java on small devices with scarce resources. In Section 3.1 we motivate the use of Java. Section 3.2 gives an overview on the Java edition for small devices. The final section, Section 3.3, is about the choice of JVM.

3.1 Why Java?

There is quite a strong trend towards Java in small handheld devices and JVMs can be expected to become available on most platforms. Currently, in early December 2002, Nokia has fifteen cellular phone models with Java [22]. Sony Ericsson has signed an agreement with Sun about licensing Java and has introduced three Java phones [29].

One advantage of using Java is its platform independence. All that needs to be pre-installed on the client device to run the Muilet is a JVM. This is Sun's vision of "write once, run anywhere".

Other advantages are that Java is object-oriented and that it has been designed to be a safe language. The access to platform details such as the file system is controlled by security managers or other safety mechanisms. Java is strongly typed, which means that the type of every variable and every expression is known at compile time. This reduces the risk of unexpected behaviour at run time. The memory management is automatic, and handled by a garbage collector. This avoids the safety problems of explicit deallocation that can be found in C and C++. Unsafe constructs, such as array accesses without index boundary checking, have been omitted from the language [9].

3.2 J2ME and MIDlets

The Java 2 Platform, Micro Edition (J2ME) is an edition of the Java technology intended for small, resource-constrained devices. It exists beside two other editions: Java 2 Platform, Standard Edition (J2SE) and Java 2 Platform, Enterprise Edition (J2EE). J2SE is targeted mainly at desktop computers, while J2EE is for server applications. These editions differ in what libraries they support for application programming and, in the case of J2ME, also in the language features supported and in the functionality of the JVM.

J2ME spans a large range of different device capabilities, from quite powerful devices like TV set-top boxes down to devices with more limited memory and processing power, like cellular phones. This has led to the concept of a *configuration* [31]. A configuration in J2ME defines a minimum platform for a category of devices with similar capabilities: the subset of Java programming language features supported[1], the functionality of the configuration's JVM, and core platform APIs, such as APIs for networking, security and installation. Two J2ME configurations have been defined: Connected Device Configuration (CDC) and Connected, Limited Device Configuration (CLDC). CDC is for devices at the upper end of the spectrum, with at least a few megabytes of memory for the storage of the JVM and libraries. CLDC is for smaller devices with as little as 160 kB to 512 kB of available memory [30]. Cellular phones are clearly targeted by CLDC while handheld computers are at the border between CDC and CLDC. CLDC is the configuration used in this thesis.

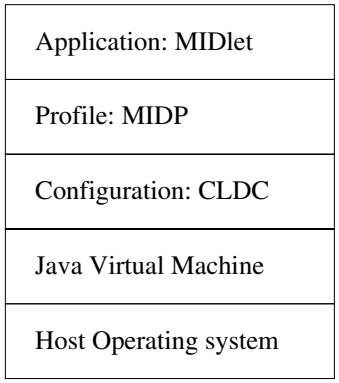

Figure 3.1: *Layers in J2ME.*

A configuration is closely coupled to a Java Virtual Machine (JVM). The configuration lies on top of the JVM (see Figure 3.1). The original JVM for CLDC is the KVM from Sun. KVM is discussed in more detail in Section 3.3.

[1] E.g., CLDC, the configuration used in this thesis, has no floating-point support.

CLDC alone is not enough to be able to write useful Java programs for a handheld device. Most importantly, it contains no user interface libraries. These are instead covered by a *profile*. A profile in J2ME groups devices with similar usage. It is more concrete and specific than a configuration, and it is implemented on top of a configuration, as seen in Figure 3.1. The interesting profile here, and the only CLDC profile implemented so far, is the Mobile Information Device Profile, MIDP [32]. MIDP targets devices with at least 32 kB of memory available for the Java heap, a screen at least 96 pixels wide and 54 pixels high, and minimum input capabilities of a 0-9 keypad, arrow keys and a select-button. MIDP devices must also have some kind of network access [35]. These demands are met by many cellular phones. Handheld computers can also run MIDP. They might not have all the necessary buttons, but a handheld computer can emulate them on its relatively large screen.

MIDP applications are called *MIDlets*. They can use features of both MIDP and CLDC. The limited environment puts a number of restrictions on MIDlets that do not apply to J2SE programs. J2SE's security model, with a security manager that gets called every time access is needed to a protected resource, is inefficient. Instead, a "sandbox" security model is used—the MIDlet runs in a closed environment, where all available APIs have been decided upon when building the JVM. There is no support for calling native functions from MIDlets, except for native functions that are linked into the JVM at compile-time. Other restrictions on MIDlets are that there are no floating-point numbers and that the error handling with exceptions is limited [30].

MIDlets are packaged in JAD/JAR file pairs. JAR (Java Archive) files are archives where the MIDlet's class files, and possibly other files like image files needed by the MIDlet, are stored together. The JAR file also includes a manifest file with the MIDlet's name, version, icon and other information about the MIDlet. This information is also present in the JAD (Java Application Descriptor) file[2]. The reason for having the information in two files is to facilitate the distribution of MIDlets over wireless networks. These networks often have limited bandwidth, and the idea is to first download the small JAD file to get some information about the MIDlet. The user can then decide if he wants to download the JAR, which is much bigger.

3.3 Choosing a Virtual Machine

Before implementation could start, a suitable JVM had to be found. There are many JVMs on the market but only a few were investigated and evalu-

[2] Actually, some types of information apply only to JAR manifest files and some only to JAD files, but the most important ones like MIDlet name and version appear in both. See [35] for details.

JVM	Dynamic loading of MIDlets	Low-level system calls	Graphical libraries	Portability	Development tools
KVM	No	No	MIDP	Yes	J2ME Wireless Toolkit
J2ME for J9	No	No	MIDP	Yes	IBM WebSphere
Java Xtreme for J9	Yes	Yes	Palm OS API	No	IBM WebSphere
IVM	Yes	Yes	Limited MIDP	Yes	Compiler only

Table 3.1: *A comparison between different Java Virtual Machines.*

ated. These are available for use on the Palm handheld, which is the client device used in our prototype implementation. More on the implementation can be read in Chapter 5.

Important factors of the compared JVMs are summarised in table 3.1. Initially, we intended to write the MuiDiscovery component (see Section 2.2) in Java. If that was the case then we would have needed dynamic loading of MIDlets from Java, and access to low-level system calls to set up wireless communication. This is the reason why these factors are in the table. As we could not find a suitable JVM which fulfilled these requirements, we eventually decided to implement MuiDiscovery in C. We also compared portability of the written programs, if a graphical user interface was supported and which development tools were available. Discussions of the JVMs follow.

KVM: *Kilobyte Virtual Machine*. The name Kilobyte reflects that KVM's memory footprint is measured in kilobytes rather than megabytes [31]. KVM is used in Sun's implementation of MIDP for Palm OS. Development is carried out in the tool J2ME Wireless Toolkit from Sun. It is a simple but robust tool without an integrated editor and without advanced debug capabilities.

J2ME for J9: J9 is a virtual machine from IBM. It is designed to run on a large number of platforms. There is a version for Palm OS which supports MIDP [14]. J9 is supported by a powerful development environment, IBM WebSphere. It has advanced debug capabilities. However, after a short period of evaluation we found several bugs in the MIDP implementation. The most severe bug was that HTTP connections did not work.

Java Xtreme for J9: J9 for Palm also supports Java Xtreme [13]. Java Xtreme is not J2ME compliant. It is a Java class library which gives access to

all OS specific functions on the Palm. This makes Java Xtreme very powerful. Everything that can be done in a native Palm application can be done with Java Xtreme. The API in Java Xtreme works as wrappers around the Palm OS API. This makes the code look like C code and programs written in Java Xtreme are non-portable.

IVM: IVM is a JVM developed by Anders Ive at the department of Computer Science at Lund Institute of Technology [11]. BlueCell has used the IVM in an implementation of JayTalk on PocketPC (for more information about JayTalk, see Section 7.2). During this work, the IVM was adapted to J2ME. IVM has been ported to Palm OS by Magnus Landqvist under the name Japalm [15]. There is support for low-level system calls in the sense that we could add our own native functions in the IVM source code and recompile the IVM. IVM is an implementation with really small memory requirements. However, parts of the MIDP/CLDC libraries have not been implemented and the graphical support is limited.

We decided not to use Java Xtreme for J9 because of its lack of portability, as we do not target the Palm only. The IVM was considered, but we figured that it would take too much time to extend the graphical libraries to fit our needs. KVM and J2ME for J9 are very similar. The bugs that we found in J2ME for J9 settled the matter. HTTP connections were crucial to our solution (see Chapter 5), so we had to use the KVM.

Chapter 4

User Interfaces for General Handheld Devices

One of the big problems studied in this thesis is to create a graphical user interface (GUI) that works on different handheld devices. We want our remote control to work on almost all handheld computers and cellular phones, with the use of Java.

The problem is that handheld devices can have very different screen sizes and input mechanisms. The challenge is to create a user interface that works on many devices and still uses as many of the features of each device as possible. Ideally, the programmer should not be bothered by these details. There should only be one description of the user interface, and not one for each type of device.

Handheld computers generally have a large screen, compared to other handheld devices. They usually have a *stylus*, a plastic pen, which is used as a mouse by pointing and clicking on the touch-sensitive screen. There are often only a few physical buttons on handheld computers. Cellular phones, on the other hand, usually have a smaller screen, which is not touch-sensitive. They have many buttons, typically a numerical keyboard, arrow keys and a select button.

The characteristics of handheld computers favour user interfaces with buttons and similar GUI elements on the screen. The larger screen enables presentation of more information at the same time. An example of a user interface on a handheld computer is found in Figure 4.1.

On the cellular phone, the smaller screen forces the user interface designer to split the information into several screens. The lack of on-screen input leads to user interfaces where the user navigates using arrow keys and the select button. Hierarchical menu systems are typical on cellular phones. See Figure 4.2.

In the next section, a discussion about Java GUIs on desktop computers is given as a background. Then follows a section which discusses existing

Figure 4.1: *Example of user interface on a handheld computer. Much information is presented on one screen, and several on-screen buttons are available.*

Figure 4.2: *Example of a typical user interface on a cellular phone. The user interface is organised in a hierarchical menu system.*

solutions to the problem of making user interfaces for general handheld devices. The last section is about our proposed solution to the problem.

4.1 Java GUIs on Desktop Computers

The Abstract Window Toolkit, AWT, is a Java library that gives the programmer a platform-independent graphical user interface. AWT has been designed for desktop computers, with a large screen and a lot of memory and processing power. The platform independence is achieved partly by the fact that the programmer does not have full control, at the pixel level, of how the GUI will be drawn [12]. An example of an AWT GUI is shown in Figure 4.3.

Figure 4.3: *An example of an AWT GUI. The top button row and the left button column are two containers that contain three buttons each. However, the two containers use different layout managers that have different strategies in how to place the buttons.*

Components and *containers* are important concepts in AWT. A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Containers are components that can contain other components. A *frame* is a top-level component, representing a window. The GUI is built by adding components to the frame and to other containers inside the frame.

The placement of components on the screen is done with the help of *layout managers*. Each container has a layout manager that decides how components should be placed inside that container. The programmer specifies how components should be placed relative to each other and the layout manager determines the exact screen coordinates.

Events are used to respond to user actions. When the user clicks a button, a Click event is sent to the *listener* that listens to events for that button. The listener can then perform some action.

Today, the standard Java GUI library is Swing. Swing is a more complex library, implemented on top of AWT. The basic principles of Swing are the same as in AWT.

4.2 Existing Solutions

As stated above, the problem we are trying to solve is to construct a user interface that works with many handheld devices and make it look as good as possible on each device. In this section a number of existing solutions are reviewed.

4.2.1 MIDP

As described in Section 3.2, MIDP is a J2ME profile intended for cellular phones and handheld computers. MIDP has a user interface library, called the *high-level API*, that is possible to implement on practically all modern cellular phones and handheld computers. The high-level API can be implemented in different ways on different devices. For example, there is an abstract user interface element called a *command*. It represents a user action. One example could be a back command. On a handheld computer this could be implemented as a button on the screen which says "Back". Cellular phones often have a physical cancel or back button. This button can be used to implement the command. There is also a possibility that the command can be put in a menu. When using the high-level API, the MIDlet programmer has little control over the look and feel of the program. You get a user interface that will always work, but will often not use the full potential of a handheld computer—the user interface will often look quite primitive. Figure 4.4 shows an example of how a high-level API user interface might look on three different devices. For more information about MIDP user interfaces, see [35].

MIDP also offers the opportunity to paint on the screen at pixel level. This is referred to as the *low-level API*. The advantage is that the programmer has full control over the appearance on the screen. Only primitive drawing operations are available, like drawing lines, circles or individual pixels. No complex objects like buttons are available. When the programmer constructs a user interface using the low-level API, he has to consider different screen sizes. It can be hard to write a program that both works well and looks good on all devices.

4.2.2 kAWT

kAWT is a project with the aim to provide a simplified version of AWT for KVM[1]. It is advanced and has a lot of the features that AWT has. The

[1]The "k" in kAWT is the same as the "K" found in KVM, standing for kilobyte.

Figure 4.4: *High-level API user interface on three different devices. The device to the left is a minimal phone with a tiny screen. The back command is implemented with a designated button. The phone in the middle has a larger screen, where more information can be presented. The back command is implemented with a button that temporarily serves as the back button. The right figure shows how the user interface would look at a Palm handheld. The back button is now implemented with an on-screen button.*

look of kAWT at a Palm handheld is compared to how the corresponding user interface would look in AWT in Figure 4.5. The problem in our case is that kAWT is poorly fitted for devices with a small screen and no stylus. The complex elements require a lot of space and on-screen buttons rely on a touch-sensitive screen. This means that kAWT suits handheld computers best. We wanted something that works on cellular phones as well, so kAWT was not suitable for our needs. Another problem is that kAWT is relatively slow.

4.2.3 UIML

The User Interface Markup Language, UIML, is a language for describing a user interface that is to be shown on devices with different kinds of user interaction. UIML targets all kinds: GUIs, text-based user interfaces, voice interfaces, speech recognition, and more—even types of interaction not yet available, e.g. virtual reality and mechanisms to receive input from brain waves. The language is developed at Virginia Tech and at Harmonia, Inc. [37].

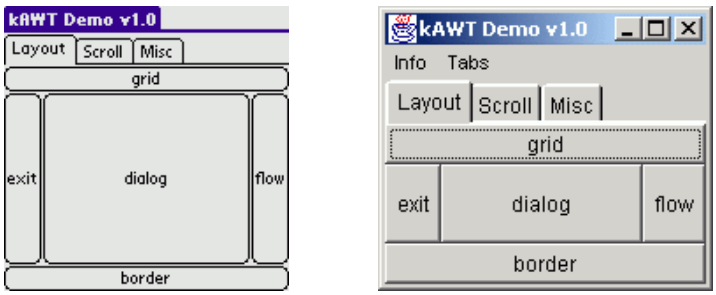

Figure 4.5: *The left figure shows an example of kAWT in a Palm handheld. The right picture shows how the same program would look on a desktop computer running Windows.*

UIML is a declarative language, based on XML[2]. The user interface description typically resides on a server. As a client requests the user interface, it can either be compiled on the server before downloading or interpreted on the client. It can first be compiled into a high-level language, such as Java or C++, and then into executable code. Interpretation requires a program on the client, capable of rendering the user interface, in the same way a web browser displays HTML.

The UIML designers have put a sharp line between the user interface and the internal logic of the application. This makes it possible to have multiple user interfaces for the same application, with little extra work.

Different elements of UIML cover different aspects of a user interface. *Structure* elements contain an enumeration of the parts that make up the interface, with abstract names for the parts. *Content* elements contain the actual information that is given to the user, e.g. text, images or sound. Input from the user is handled by *behaviour* elements. They are rule-based systems stating what should happen in response to different events. The elements that are most interesting in this context, when it comes to the adjustment of the user interface to different target platforms, are the *style* and *peers* elements. Together, they specify a mapping from abstract interface parts or classes of interface parts to constructs in a target language. E.g., the abstract name "Command" might be mapped to the Java class `java.awt.MenuItem`, if Java is the target language. Then, commands are shown as items in a menu. A UIML fragment that shows this mapping in a peers element is given in Figure 4.6. There is one mapping for each target device or family of target devices. The mappings have to be specified explicitly by the UIML programmer [1].

To use UIML in our project, we would have to create an interpreter on the Palm handheld and transfer the GUI as UIML code from the server.

[2]The Extensible Markup Language, XML, is a markup language and uses tags that look like HTML tags, but XML is intended to be a format for storage of any information, not just web pages.

```
<uiml>
  ...
  <peers>
    ...
    <presentation name="Java-AWT">
      <component name="Command" maps-to="java.awt.MenuItem"/>
    </presentation>
  </peers>
  ...
</uiml>
```

Figure 4.6: *A UIML fragment with a mapping from the abstract name "Command" to the Java class* `java.awt.MenuItem`.

At the UIML home page there is a Java renderer available that can interpret UIML code and display it as AWT or Swing GUIs [37]. It would certainly be possible to implement such an interpreter on the Palm, adjusted to MIDP. Still, we concluded that it would require too much work. We continued with the idea of transferring a MIDlet instead of transferring UIML code.

4.3 Our Library: `muigui`

We have constructed a solution that is a combination of the high-level and low-level MIDP APIs. We have built a graphical library, `muigui`, that adapts to the available resources. If the device has sufficient resources in terms of screen size and stylus, the low-level API is used and an appealing user interface is drawn. This is called the graphic mode. Otherwise, the high-level API is used to guarantee the functionality of the user interface. This is the menu mode of `muigui`. The programmer does not have to worry about these details and can use the library as if a large screen and a stylus were always available. Naming of classes and methods in the library is from that perspective. Adaption to the menu mode will be made below the surface if it is needed. In appendix B, short descriptions of the classes in `muigui` are given.

`muigui` has been designed to be an even more simplified version of AWT than kAWT. Only a small subset of the components is available, but the concepts and ideas come from AWT. Components are placed inside containers in the same way as in AWT and a layout manager decides their exact placement. If the layout manager concludes that it is impossible to place all components within the screen, or if there is no stylus, the transition to the menu mode is made. Then, menus are created which follow

the hierarchical structure of the containers and components. An example of this is found in Figure 4.7.

Figure 4.7: *The left and middle figure show a* `muigui` *user interface in menu mode. The right figure shows the same user interface on a Palm handheld in graphic mode. In the menu mode the eight commands are separated into three screens accessed from the main screen shown in the left figure. The middle figure shows one of these screens.*

`muigui` has been made with the assumption that it will be used in a MIDlet that communicates with a server. Commands can be attached to buttons and other elements. If a command has been attached, it will be sent to the server automatically when the button is pressed. An example of how `muigui` is used follows. It is the same program as shown in Figure 4.7.

```
..

public class SonyVideo extends MIDlet implements CommandListener,
                          ButtonListener {
    public SonyVideo() {
    super();
    screen = new MuiScreen("Sony Video SLV-SE220", this);

    // Create the CommandHandler
    CommandHandler commandHandler = new AVCommandHandler();

    // Create the Panels
    Panel panel1 = new Panel("System", false);
    Panel panel2 = new Panel("Operation", false);
    Panel panel3 = new Panel("Stop/Pause", false);
```

```
    // Create layoutManagers
    SimpleLayout layout1 =
        new SimpleLayout(SimpleLayout.ROW_LAYOUT);
    panel1.setLayoutManager(layout1);

    ...

    // Create the buttonImages
    ButtonImage bImagePlay = new ButtonImage("/Play.png",
                         "/Play.png");
    ...

    // Create the buttons
    Button buttonPlay =
        new PrettyButton("Play", AVTarget.OPID_PLAY, commandHandler,
                 PrettyButton.BUTTON_MODE_IMAGE_FRAME,
                 bImagePlay);

    ...

    // Add the buttons to the panels
    panel1.addComponent(buttonOnOff);
    panel1.addComponent(buttonEject);

    ...

    // Set the layout for the screen and add the panels
    screen.setLayoutManager(new SimpleLayout(SimpleLayout.COLOUMN_LAYOUT));
    screen.addComponent(panel1);
    screen.addComponent(panel2);
    screen.addComponent(panel3);

    // Prepare the screen for showing
    screen.pack();
    }
    ...
}
```

More information about `muigui` can be found in Appendix B. The constant `AVTarget.OPID_PLAY` defines the command that will be sent to the server when the button is pressed. AVTarget is a class that handles communication with the server according to the Bluetooth profile AVRCP. More information about AVRCP can be found in Section 5.6.

To sum up, `muigui` is an attempt to combine the portability of MIDP and the more graphically appealing user interfaces of AWT.

Chapter 5

Implementation

In Chapter 2, we discussed the architecture of Mui at a generic level, stating components that the devices should contain and how the components should interact. In this chapter, the implementation of the components in a prototype system will be described.

For the prototype, a consumer electronics application has been chosen. A Sony VCR is controlled from a Palm handheld, using Bluetooth. A Java application with a GUI is downloaded from the VCR to the handheld, which can be used as a remote control.

The reason for choosing a handheld computer is that it is easier to program than, e.g., a cellular phone. Programming a cellular phone would require special equipment not freely available. Among the different handheld computers on the market a Palm handheld was chosen, mainly because it has good Java support in the form of the KVM and other JVMs, and Bluetooth is well supported with an expansion card. Palm, Inc. is the market leader in handheld computers, so Palm experience was interesting for BlueCell. We chose the VCR since it seemed relatively easy to connect to and control. The choice of Bluetooth was natural, because Bluetooth has built-in mechanisms for discovering other devices and communicating with them in wireless networks. Furthermore, BlueCell had extensive experience from working with Bluetooth in earlier projects and we could get good support from our supervisors at BlueCell. For more information about Bluetooth, see Appendix C.

When building the prototype, we did not have access to optimal equipment for all parts. For some parts solutions had to be chosen that were slower and perhaps less elegant than we would have liked.

In the next section, the hardware used in the implementation will be presented. Sections 5.2 and 5.3 refer to the architecture in Chapter 2 and discusses how its different parts have been implemented. The last sections, 5.4 to 5.6, describe the implementation in more detail.

5.1 Description of the Hardware

This section presents the hardware in our implementation. Figure 5.1 shows the hardware components, and the rest of this section describes them.

Figure 5.1: *Hardware components in the system. In the top row are the components in the server device. From the left: the Wireless Information Provider (WIP), the Programmable Logic Controller (PLC) and the VCR. The small black box above the PLC contains an electric circuit built by BlueCell to connect the PLC to the VCR. The client device is the Palm handheld with a Palm Bluetooth Card, in the lower left corner.*

5.1.1 Palm Handheld

A Palm m515 has been chosen as handheld device. It is a fairly new handheld computer with colour display (see figure 5.2). Palm, Inc. has made many models with similar appearance. They have an operating system called Palm OS, of which our handheld uses version 4.1. Palm, Inc. has a focus on usability. A first-time user should be able to get started quickly.

The Palm m515 screen is touch-sensitive. The user has a stylus, a small plastic pen, which is used as a replacement for both a mouse and a keyboard. One area on the screen, the Graffiti area, is designated for text input. Here, the user draws characters with the stylus, which are recognised by the operating system and sent to the current application.

Since the Palm handheld has limited memory and is battery-powered, the programmer has to constantly keep the memory requirements in mind. The battery can be charged in a cradle, which is connected to a desktop computer. The cradle is also used to install programs, through a process

called HotSync. There is an IR port for beaming programs between Palm handhelds. For more information about Palm handhelds, see [25].

Figure 5.2: *Palm m515.*

5.1.2 Palm Bluetooth Card

Palm Bluetooth Card is an expansion card, which can be used to get Bluetooth capabilities in a Palm handheld (see Figure 5.3). The card can be used to discover other Bluetooth nodes and establish connections. There is an API for accessing the Bluetooth stack.

Figure 5.3: *Palm Bluetooth Card.*

5.1.3 Wireless Information Provider

The Wireless Information Provider 125 (WIP) is a product from ConnectBlue [5] (see figure 5.4). It is used to give wireless access to industrial equipment via a web interface. The WIP is Bluetooth-equipped and runs a web server from GoAhead, which includes an HTTP server and an FTP server. The communication with the equipment is done via the Modbus

protocol, a standardised protocol for controlling industrial equipment. For more information about Modbus, see [18].

Clients access the WIP by opening a wireless network connection and are then able to download web pages from the HTTP server. The HTTP server supports ASP (Active Server Pages) scripts. ASP is a technology from Microsoft for creating dynamic web pages. When a client requests an ASP page, a script is executed on the server. This script produces output, normally in HTML, which is sent back to the client and typically shown in a web browser. In the WIP, the ASP scripts can communicate with the industrial equipment through a WIP specific extension of the script library.

Figure 5.4: *Wireless Information Provider 125.*

5.1.4 Programmable Logic Controller

A Programmable Logic Controller (PLC) is a device used to automate monitoring and control of industrial equipment. We have used a product from Schneider Electric, called PLC TSX 07 (see figure 5.5). It has nine digital inputs and seven digital outputs, and can communicate via the Modbus protocol. It can be programmed in an assembly language, with the development tool PL7-07, to put signals on the outputs in response to signals on the inputs and to Modbus data.

5.1.5 Video Cassette Recorder

We have chosen a video cassette recorder, a VCR, as the device the user can control. The model is Sony SLV-SE220D. It is a standard VCR with basic functionality (see figure 5.6). Some modifications have been made

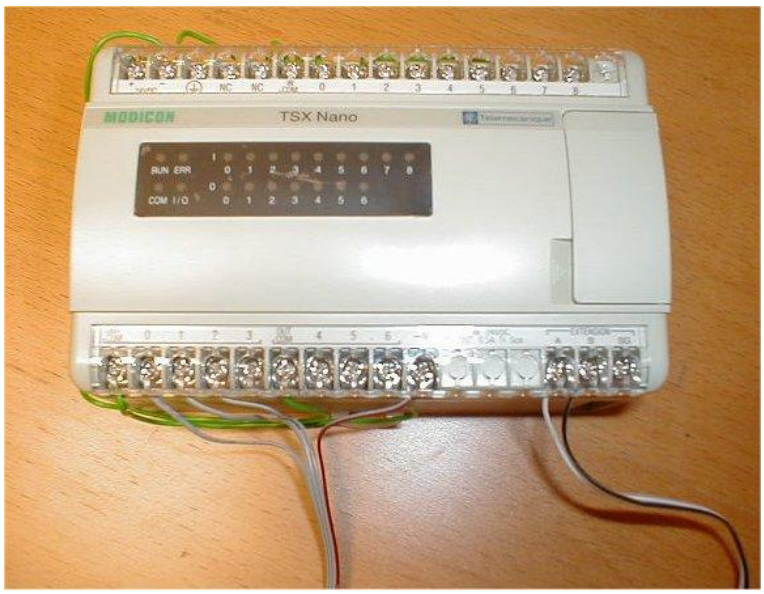

Figure 5.5: *PLC TSX 07.*

to the VCR, so it can be controlled from the PLC, but the VCR can still be operated in the normal way. The modifications will be explained in section 5.5.

Figure 5.6: *Sony SLV-SE220D.*

5.2 Components in the System

The layered view of the architecture from Chapter 2 is repeated in Figure 5.7. Figure 5.8 shows the components in the implementation.

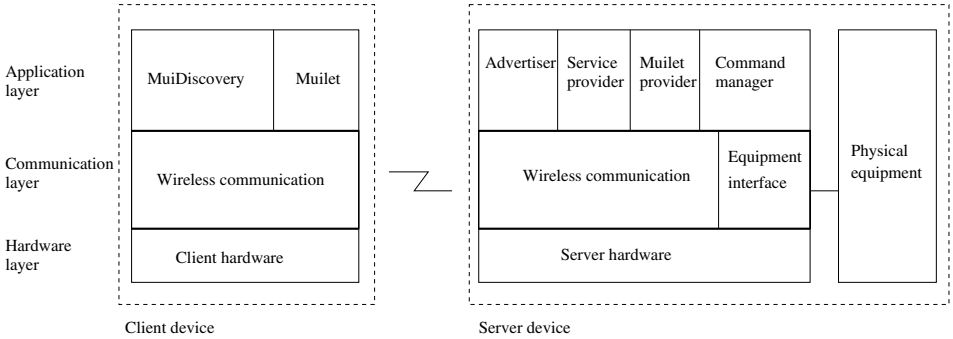

Figure 5.7: *A layered view of the architecture, repeated from Chapter 2.*

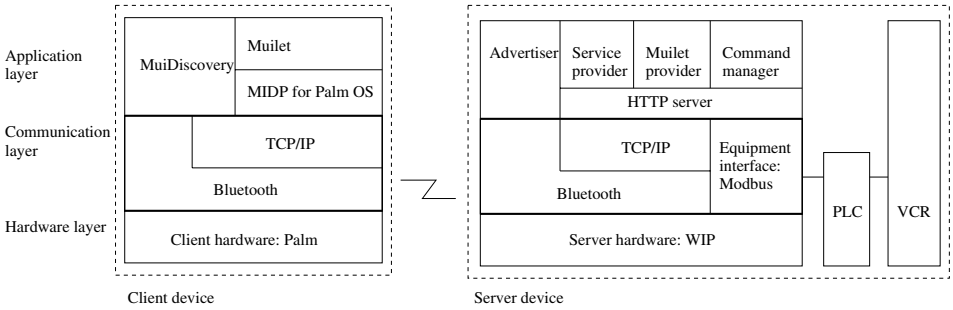

Figure 5.8: *The implementation of the architecture. Shows how the components in the system have been implemented.*

The client hardware is the Palm handheld, and the WIP acts as server hardware. The wireless communication is implemented by Bluetooth, where the Bluetooth capabilities in the Palm are given by the Palm Bluetooth Card, while the WIP has a built-in Bluetooth chip. TCP/IP enables ordinary Internet network services such as HTTP for upper layers. The HTTP server is the existing one in the WIP, which was delivered as part of the product.

On the client device, the MuiDiscovery component has been implemented as a Palm OS C application. Java was not used here, because we needed the possibility to download and start MIDlets, and access to the Bluetooth stack. The Muilet component is a remote control MIDlet, which has been implemented using our `muigui` package. As discussed in Chapter 3, Sun's MIDP for Palm OS with the KVM has been chosen as runtime environment for the MIDlet.

The Advertiser component in the server device is implemented by using the inquiry mechanisms of Bluetooth. The HTTP server is used to implement three components: Service provider, Muilet provider and Command manager. The Service provider is implemented by keeping an XML file on the HTTP server, listing the services. The Muilet provider supplies the remote control interface as a MIDlet file on the HTTP server. The Command manager is implemented as ASP scripts running on the HTTP server. The equipment interface is the Modbus protocol, mentioned in Section 5.1.3—the Command manager communicates with the VCR via Modbus. The PLC translates the Modbus signals into signals understood by the VCR.

5.3 Flow of Events

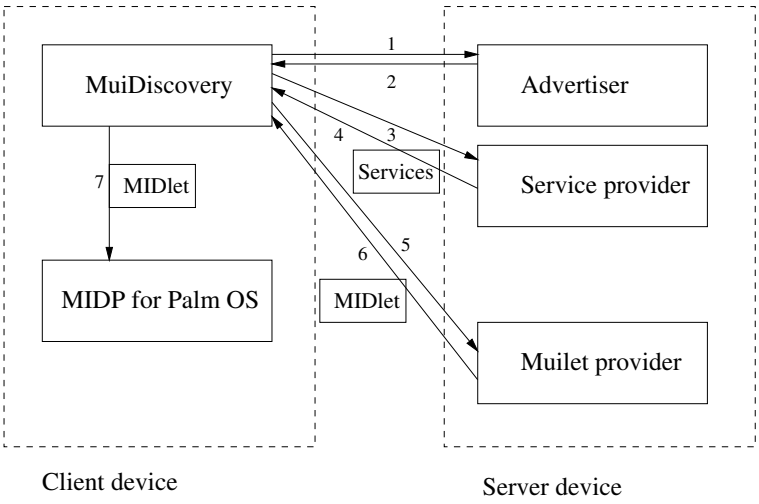

Figure 5.9: *Flow of events for the use-case "Discovering and starting a service" (see Section 2.3).*

In this section, we discuss the same use-cases as in Section 2.3, but this time from an implementation perspective. For the use-case "Discovering and starting a service", shown in Figure 5.9, events occur in the following order:

1. MuiDiscovery uses the inquiry mechanisms of Bluetooth to discover devices within range. The server device is found.

2. Advertiser responds and a wireless connection is established.

3. MuiDiscovery asks for a list of services from the Service provider. This is done as a request for an XML file from the HTTP server. The Mui services are listed in the XML file.

4. The XML file is sent back to the Palm handheld and the list of services is shown to the user.
5. The user selects a service by tapping its icon, and MuiDiscovery asks the Muilet provider for the service's MIDlet file. This is done as an HTTP request.
6. The MIDlet is downloaded from the HTTP server and installed on the Palm handheld.
7. MIDP for Palm OS is started and runs the MIDlet.

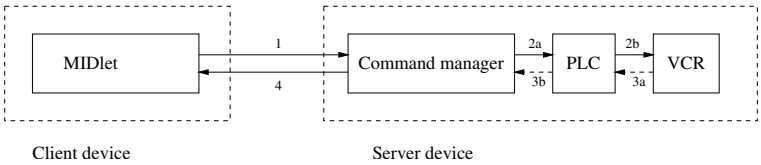

Figure 5.10: *Flow of events for the use-case "Using a service" (see Section 2.3).*

The steps for the use-case "Using a service" continue where the first use-case ends. This use-case is shown in Figure 5.10.

1. The user taps a button on the remote control. A command is sent to the Command Manager. This is done by requesting an ASP page from the HTTP server. The command is embedded in the request as a parameter in the URL.
2. (a) The ASP script executes on the HTTP server. Signals are sent to the PLC via Modbus.
   (b) The PLC reacts to the signals from the WIP and controls the VCR.
3. Handling responses from the VCR is not implemented. There is no response from the VCR, and thus no need for the Command manager to wait for information from the PLC.
4. The ASP script sends a response to the Palm handheld.

5.4 MuiDiscovery

In this section, the implementation of MuiDiscovery is described. MuiDiscovery is an application that runs on the Palm handheld. It is started by the user, when he wants to use nearby Mui services. MuiDiscovery finds services and shows them on the screen, where the user can select a service by tapping on the service's icon. MuiDiscovery then downloads the MIDlet, installs it on the Palm handheld and starts it.

MuiDiscovery has been implemented in C since it gives a high degree of control and access to all system calls and to the Bluetooth stack. As discussed in Section 3.3, we did not use Java for MuiDiscovery, because we would not have been able to install MIDlets or access the Bluetooth stack if we did. The implementation was done in Metrowerks CodeWarrior for Palm OS Platform, a development environment which allowed us to debug directly on the Palm handheld. More information about Metrowerks CodeWarrior can be found at [17].

One of the main goals in the implementation was to provide a user interface that was very easy to use. The number of steps the user has to go through should be minimised. All details about Bluetooth, such as names of Bluetooth nodes, are hidden from the user and the services are shown directly.

The following subsections describe different parts of the implementation.

5.4.1 Discovering Services

Services are discovered at the start of the execution of MuiDiscovery. These are not updated spontaneously, but only at the user's initiative. The inquiry functions in the Bluetooth API are used to discover all Bluetooth nodes in the neighbourhood[1]. The first task was to distinguish those nodes that support the Mui protocol. Here, problems occurred. The only information that could be retrieved about a node was a 48-bit Bluetooth device address and the *friendly name* of the node as a string. The friendly name of a Bluetooth node is a descriptive name that is intended to be displayed to the user. It has a maximum length of 249 characters, but names longer than 20–30 characters are not suitable for display on a small screen, such as the Palm handheld's.

The only way, with our equipment, to get more information about a node, besides the device address and the friendly name, would be to open a TCP/IP connection and ask for some specific Mui file[2]. This is not very practical, and would take a long time if the device was not a Mui device. Therefore, it was decided to put the information in the friendly name; if the friendly name of the node begins with "MuiServer", MuiDiscovery assumes that it is a Mui device. This is not an optimal solution, though, and a discussion about other solutions can be found in Chapter 6.

When MuiDiscovery has verified that a node is a Mui device, a TCP/IP connection is established to the device. This caused some problems during the implementation:

[1] Inquiry is handled by the baseband layer, below the Host Controller Interface (HCI) in figure C.1 of Appendix C.

[2] TCP/IP connections are possible, because the Bluetooth LAN access profile is used. This profile is supported by the WIP and by the Palm Bluetooth Card. Bluetooth profiles are discussed more in Appendix C.

- The Palm OS network library, NetLib, only allows one open connection at a time. During connection and download, the Palm handheld is locked and the user cannot interact with the device. Therefore, it was not possible to connect in the background, hidden from the user. This can make it a bad experience for the user.

- Palm, Inc. has designed NetLib so that for all network connections, the user should first make a configuration manually and then select it using the Palm handheld preferences [39]. The ambition was to automate this, because services should just pop up when found. We got help from a newsgroup to solve one part of the problem [4] and the rest of the problem was solved by manipulating Palm OS internal databases for network services. This solution will probably not work in other versions of Palm OS than 4.1.

- The Bluetooth connection sometimes does not work, even if it is correctly configured. This happens even if the user does it the way Palm, Inc. suggests, from the Preferences panel. We believe this is due to a bug in the Bluetooth stack. The only solution found is simply to try again.

The next step, after establishing a connection, is to find out which services the Mui device offers. The services are described in an XML file that is downloaded with HTTP. The library XMLlib is used to parse the XML code (for more information about XMLlib, see [36]). All available services are shown to the user in a tree structure. Icons for the services are also downloaded. See figure 5.11.

5.4.2 Downloading and Installation

When the user taps a service in the list, a MIDlet is downloaded from the server via HTTP. Here, an unexpected problem was encountered. If the file was too large, larger than 1346 bytes, the transfer was always interrupted prematurely. Our theory is that only files that fit into one Ethernet frame, including the added HTTP header, could be downloaded due to a bug in Palm OS. A lot of time was spent trying to solve the problem before abandoning the idea and deciding to use a workaround. The MIDlet is split into several small files, all small enough to fit into one package, and all are put on the server. The splitting is done with a tool we created. Since a new HTTP request has to be done for every file, and a new TCP connection has to be established for every HTTP request, this leads to slow transfer. E.g., the transfer of a 68 K MIDlet takes around one minute. This is of course unacceptable in a real application.

The format of the stored MIDlet on the server also led to trouble. We wanted to store it as a JAR file (see Section 3.2) and install the JAR file on the Palm handheld, since JAR files are platform independent. However,

Figure 5.11: *List of services shown on the Palm handheld.*

MIDlets for Sun's KVM on the Palm are stored in PRC files. The PRC file format is the format all Palm applications are stored in. The reason for having MIDlets in PRCs is that this makes it easy to install them via the HotSync mechanism, like other Palm applications. One solution would be to convert the JAR file to a PRC file in the Palm after downloading. This was beyond the scope of this thesis, though, so it was decided to store the MIDlets as PRC files on the WIP, which makes the implementation Palm specific. A JAR file can still be put on the server to support other client types.

The MIDlet is installed on the Palm handheld as an ordinary Palm application as it downloads.

5.4.3 Starting the MIDlet

Since the MIDlet is installed as an ordinary Palm application, it was easy to start it from MuiDiscovery, with use of the Palm OS API. After downloading the MIDlet once, the user can also start it himself in the same way he starts other Palm applications: from the Application Launcher, which is the Palm OS counterpart to the Microsoft Windows desktop. The MIDlet remains installed after download, and it can be synchronised by HotSync in the same way as other applications.

5.5 Server Device

This section discusses the implementation of the server device. The hardware components were described in Section 5.1. The server device is made up of the WIP, the PLC and the VCR.

The WIP was chosen because it is an existing product which is very stable, and we could get good support on it. The HTTP server offers storage and retrieval of the MIDlets and the XML file with services. The Service Provider and Muilet Provider components were implemented as files stored on the HTTP server.

The Advertiser component was implemented with the inquiry mechanisms of Bluetooth, as discussed in Section 5.4.1. This required no modification on the WIP, apart from changing its friendly name to start with "MuiServer".

The WIP is built to control equipment with the Modbus protocol. Modbus and ASP scripts were used to implement the Command manager component. The MIDlet makes an HTTP request for an ASP page on the WIP. Input to the Command manager, i.e. what command should be executed, is given as parameters in the ASP URL. In Figure 5.12, a fragment from the ASP code is shown, explaining what happens when the user issues a Play command. The response is encapsulated in the HTML output from the script, and extracted by the MIDlet on the client side. The resulting HTML output from the Play command in Figure 5.12 is also shown in Figure 5.13.

The disadvantage with using the WIP in this project is that it only offers control over the equipment through ASP scripts. This slows down the whole process, since a new HTTP connection has to be established for every command. We would have wanted to write our own software in the WIP running as a server daemon[3]. Then, we could have defined our own protocol and one connection could have been established and kept open during the entire life of the MIDlet. The WIP did not offer this flexibility, so it was not an option. More is said about alternative solutions in Chapter 6.

The next problem is that Modbus is suited for industrial equipment and is not used in home appliances. To connect the WIP to the VCR, the PLC was therefore used as a translator from Modbus to signals that could be understood by the VCR. The PLC was programmed with the PL7-07 tool on a PC, and the programs could be downloaded to the PLC for debugging and execution.

BlueCell helped with the task of controlling the VCR from the PLC. Several options were considered. The first option was to use the I^2C bus. This is a de-facto standard for communication between Integrated Circuits

[3] A *daemon* is a process which runs in the background and provides some service to an application, on behalf of a user. The term "daemon" is generally used in association with a service running on a Unix machine. Some daemons include web servers, telnetd, crond, and sendmail. (Definition taken from [7])

```
<html>
<head>
  <title>Command</title>
</head>
<body>
<h3>Command</h3>
<%
  // Define operation IDs
  opidPlay        = 68; // (=0x44)
  opidStop        = 69; // (=0x45)
  ...

  // Define responses
  responseAccepted =  9; // (=0x09)

  if (command == opidPlay) {
    // Write to Modbus interface
    aspGBIWriteBit("1","11","1");

    // Return ACCEPTED
    write("<mui response=\"");
    write(responseAccepted);
    write("\">\n");
  } else if (command == opidStop) {
    ...
  }
%>
</body>
</html>
```

Figure 5.12: *An excerpt from command.asp, the ASP script on the WIP that handles communication with the VCR. The part inside the* `<% ... %>` *is code that is executed, and the rest is written directly as output. When the user issues a Play command, the following URL is requested:* `http://WIP/command.asp?command=68`*. The ASP variable* `command` *gets the value 68, which corresponds to Play.* `aspGBIWriteBit` *is a WIP-specific command for sending data via Modbus. The script produces output, shown in Figure 5.13, which is read by the MIDlet.*

```
<html>
<head>
  <title>Command</title>
</head>
<body>
<h3>Command</h3>
<mui response="9">
</body>
</html>
```

Figure 5.13: *Output from the ASP script, when the user has issued a Play command. The response (in this case 9, which stands for Accepted) is extracted from the HTML data by the MIDlet.*

in consumer electronics, invented by Philips Semiconductors [26]. The plan was to connect to the bus and send commands from the PLC, but this solution was abandoned since it was too complicated.

The second option that came up was to use the existing IR port on the VCR and build an IR transmitter that would be controlled from the PLC. The IR transmitter would then be placed in front of the IR port of the VCR. This solution would have required reverse-engineering of the IR protocol of the VCR. BlueCell had done this before for another system, but it would still require a lot of work, so this solution was also abandoned when a new solution emerged, that was much more simple and elegant.

The solution was basically to connect to the back of each front panel button from inside the VCR. A pressed button was simulated by raising the voltage level for a short time. BlueCell built an electric circuit that transforms voltages from 24 V logic out from the PLC down to 5 V logic into the VCR. They added a new connector at the back of the VCR where the signals from the PLC enter (see Figure 5.14). The signals are made up of three bits. This gives eight different command values, seven of which can be used (an all zero bit has no command functions). The commands are listed in Table 5.1. Seven commands are too few in a real application, but were considered enough in this prototype implementation, as only three bits were available in the connector used. Inside the VCR, the signals are demultiplexed to seven lines and each line is connected to a button on the front panel of the VCR. Thus, the VCR can be controlled from the PLC and ultimately from the Palm handheld.

Two-way communication is a part of our concept, meaning that the user should be able to see status information about the VCR in the MIDlet. This would be possible with our equipment. A signal on the VCR could have been connected to an input of the PLC, and Modbus supports communication from the PLC to the WIP. The WIP could then give status information to the MIDlet in the form of output from the ASP script.

However, this proved to be quite complicated with the VCR chosen. We wanted to use information from the display on the front panel, e.g. showing if a tape is inserted in the VCR (see Figure 5.15). The problem was that there were no simple static signals inside the VCR representing the symbols on the display. Therefore, this was not implemented, and there is only one-way communication. This means that the answer from the ASP script to the Palm handheld is always the same.

LINE-2/LIGNE-2 (DEC./EXT.)
SONY
VIDEO CASSETTE RECORDER
MODEL NO.
~ AC 220 240V 50Hz
SONY CORPORATION
SER.NO.
0254952
SHOWVIEW is a registered trademark of Gemstar Development Corporation. The SHOWVIEW system is manufactured under licence from Gemstar Development Corporation.

Figure 5.14: *The back of the VCR. New connector added by BlueCell marked by an arrow.*

Figure 5.15: *The display on the front panel of the VCR. The symbol marked by an arrow shows that a tape is inserted in the VCR.*

5.6 Remote Control MIDlet

The MIDlet implemented in this project has been named SonyVideo. It runs in the MIDP for Palm OS runtime environment on the Palm handheld. The MIDlet shows a simple graphical user interface with buttons

Signal	Command
000	No command
001	On/off
010	Eject
011	Rewind
100	Fast forward
101	Pause
110	Stop
111	Play

Table 5.1: *VCR commands.*

that represent the different commands (see Figure 5.16). The MIDlet works as a remote control for the VCR.

Figure 5.16: *The GUI of the MIDlet. The eighth button, the Record button in the lower right corner, is not functional—it sends no command to the VCR. This is because only seven commands were available.*

The MIDlet uses the graphical library `muigui` (see Section 4.3). The large part of the work was to develop `muigui`. Then, the remote control MIDlet was less complicated to implement, and only two major issues arose during the implementation. The first problem was with the memory that made the MIDlet crash when started. The cause was that the Bluetooth stack was running and that it used some memory which normally is not used when a MIDlet is started, so the KVM tried to allocate more memory

than was available. The solution was to decrease the available memory for the KVM manually. The second issue was that the wireless connection sometimes failed. This problem was the same as the one discussed in section 5.4.1. The remedy here was to retry making the connection.

To decrease the rather long transfer times of the MIDlets, it would have been useful to pre-install the `muigui` library on the Palm. We did not succeed in doing so, and the whole library had to be included into every MIDlet. This is not optimal.

The `muigui` library has been constructed especially for GUIs that control a server device. Commands can be attached to buttons so that a command is sent to the server automatically as you select the button. In our implementation, the communication with the server has been implemented via HTTP requests.

Controlling audio/video equipment via Bluetooth has been specified in a Bluetooth profile[4], Audio/Video Remote Control Profile (AVRCP) [3]. BlueCell has made a Java API to support AVRCP. The API also complies with JSR-82, an upcoming standard for accessing Bluetooth from Java. JSR-82 is discussed more in Chapter 6. We have used this API at a high level in the Java code. AVRCP's underlying transport protocol, AVCTP, has been replaced by HTTP requests. It would be simple to switch the implementation to AVCTP when it becomes available.

In Chapter 1, it was stated that one advantage of downloading the remote control user interface from a server is that a standardised remote control protocol is not needed. Then why use AVRCP? In some ways it can be advantageous. If the server follows AVRCP, it can serve not only client devices with Mui, but also other devices that use AVRCP.

[4]For more information about Bluetooth profiles, see Appendix C.

Chapter 6

Evaluation

The focus of this project was not to find the best possible solution, but to build a prototype that works. We have not always evaluated all possibilities before buying equipment, which the schedule of this project did not allow. In this chapter, different aspects of the implementation will be evaluated and a description will be given of how we think an optimal solution would be implemented, based on our experiences from this project.

6.1 Components in the System

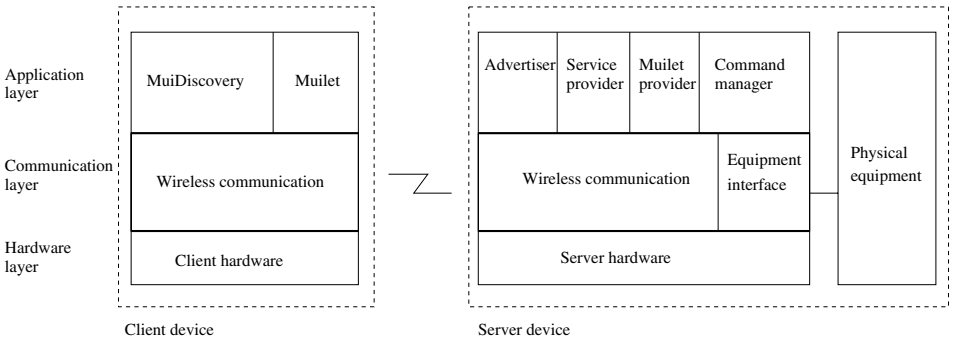

Figure 6.1: *The components in the system.*

This section contains an evaluation of the different components in the system, shown again in figure 6.1.

6.1.1 Client Device

The components in the client device have been evaluated as follows:

Client hardware: The Palm handheld was a good choice. It was easy and straightforward to program, and the documentation and help on the Internet was readily available. The drawbacks were that there was

only one thread of execution, and difficulties with network connections. The amount of long time memory, where applications and databases are saved, was sufficient. The working memory caused no problems for the C application, but there were some problems with Java.

A handheld computer is a reasonable handheld device for controlling equipment. The large colour screen on the Palm handheld made the demonstration enjoyable. It would also have been interesting to test it on cellular phones.

Wireless communication: Bluetooth worked well and we think Bluetooth is a good choice in a commercial system. Bluetooth is well suited for this kind of system and the built-in discovery of nodes is useful. There were some problems with the TCP/IP connection over Bluetooth implementation from Palm, Inc., i.e. that the connection failed sometimes and large files could not be transferred at all. This and other similar problems will probably be solved when Bluetooth on Palm handhelds becomes more mature. The problem of large files that could not be transferred was solved by sending large files in pieces. This did, however, make the transfer times for MIDlets unreasonably long, because of the repeated HTTP requests, and this would be unacceptable in a commercial system. Discovery of other Bluetooth nodes with the inquiry mechanisms of Bluetooth worked well, but was not optimal (see the text about Advertiser below).

One alternative to Bluetooth would be Infrared (IR). The Palm handheld is already equipped with an IR port and the ordinary remote control of the VCR already uses IR. This means that the VCR could be controlled directly from the Palm handheld after the MIDlet has been installed without modifications of the VCR. However, to replace Bluetooth with IR completely, the download of the MIDlet would also have to be implemented. This would require two-way communication over IR and an IR transmitter would have to be added to the VCR.

MuiDiscovery: MuiDiscovery was implemented in C since we wanted to do things not supported in Java, like accessing the Bluetooth stack and installing MIDlets on the Palm handheld. We are content with our choice of C in this project. Everything we wanted to do with the Palm handheld was, in principle, possible. It was just a matter of finding the right API function and figure out how to use it. The available documentation was generally good [24].

However, it would be advantageous to implement MuiDiscovery in Java and benefit from Java's platform independence. Access to the Bluetooth stack from Java would have been possible if the specification for Java for Bluetooth, JSR-82 [19], had been implemented in

MIDP for Palm OS. This specification is finished, but has not been implemented in Palm OS. JSR-82 offers APIs to access layers in the Bluetooth stack. Then, we could have made a non Palm OS specific implementation of the discovery of other Bluetooth nodes.

Over-the-Air provisioning (OTA) is a specification for how MIDlets can be stored on servers, retrieved over wireless networks and installed. OTA could have been used to install MIDlets on the Palm handheld from Java. In MIDP 1.0, the version of MIDP discussed so far, OTA is not part of the actual specification, but dealt with in a separate document called a Recommended Practice [33]. OTA is not mandatory in MIDP 1.0, and it has not been implemented in MIDP for Palm OS. In the next version of MIDP, MIDP 2.0 [34], OTA will be part of the specification. MIDP 2.0 currently exists only as a reference implementation from Motorola and was not available during the implementation. Thus, it was not possible to use OTA.

Muilet: The Muilet component was implemented as a MIDlet. The GUI is shown as the MIDlet runs on MIDP for Palm OS. Representing the GUI in this way, as a part of a Java application that is downloaded, is a solution that we think would work well in a commercial system.

There are alternative ways of representing the GUI. A simpler way is to represent the GUI in a more abstract format, where the elements of the GUI are listed and commands are attached to some of the elements. These commands are sent to the server as the element is activated. After downloading, the abstract description is interpreted in the client device, and the GUI is displayed. UIML, discussed in Section 4.2.3, is one such GUI description. If an abstract description is chosen, one interpreter has to be implemented for each client type, which is a drawback. The KVM, or similar JVMs, can be expected to be found on all platforms.

The GUI has been implemented on top of MIDP. MIDP 1.0 is rather limited, and we think it needs some extension for MIDlets to be able to compete with native applications, perhaps something like the `muigui` library. `muigui` has worked well and given an elegant user interface on the Palm handheld in graphic mode (see Section 4.3). The idea with two modes could be reasonable.

As discussed in Section 5.6, it would have been good if `muigui` had been pre-installed on the Palm handheld. Then the MIDlets could have been made much smaller and the transfer times would have decreased.

In the next version of MIDP, MIDP 2.0, several changes have been made in the direction of allowing more control over the look of the user interface in the high-level API, while preserving compatibility

with the most primitive devices. The most important changes are the introduction of layout on the screen and the possibility to create custom items (custom components). Layout is not handled by a separate layout manager, but directly by the Form class, which is the container for items. There is a possibility to specify minimal sizes and preferred sizes for items. These sizes are taken into account as the layout is made. Custom items are created by subclassing the CustomItem class. The programmer has control over both user interaction and appearance of the custom item.

If MIDP 2.0 had been available, the need for the `muigui` had certainly been smaller. As we have only read the documentation, and not been able to try it out in practice, it is too early to tell if MIDP 2.0 would have been enough.

6.1.2 Server Device

The components in the server have also been evaluated:

Server hardware: The WIP is not well suited to this project. A more programmable device would have been more suitable. We do not think that a solution with external Bluetooth hardware would work in the VCR case. For the product to be attractive to consumers, the Bluetooth hardware has to be integrated in the VCR. The Bluetooth circuits and other functionality in the WIP would then be inside the VCR, and the PLC would not be needed (more about this under Physical equipment below).

Still, the WIP has fulfilled its role in the project. It has been very stable and has not caused much trouble. It offered simple Bluetooth access to the VCR and we got good support from ConnectBlue.

Wireless communication: The discussion about Wireless communication for the client device also covers this part.

Equipment interface: Modbus was not a very good protocol in the project, because the VCR could not communicate with it. Since Modbus was the only option available from the WIP, the PLC had to be used as a translator. In a commercial application, with all Bluetooth circuits inside the VCR, the Equipment interface would rather be an internal bus in the VCR, e.g. an I^2C bus.

Advertiser: Our solution for Advertiser, using the inquiry mechanisms of Bluetooth and changing the friendly name of the Bluetooth node so that it starts with "MuiServer", is not really a good idea. This name is intended to be a descriptive name, a name the user can see. Technical information such as this should not be coded in this way.

If the WIP could be programmed in more detail, we would have used the Bluetooth Service Discovery Protocol (SDP) and put more information than just the name in the response from the WIP at the SDP level (see Appendix C). There, the WIP could have announced that it was a Mui server. This is the way BlueCell has done it in JayTalk (see Section 7.2).

Service provider: The solution to store the list of services as an XML file on the server was a good idea, and to use HTTP to access the file is also a good alternative. This is the same solution that is used in JayTalk. HTTP is an established way of retrieving files in distributed systems, and small HTTP servers exist that work in embedded systems, e.g. the GoAhead webserver that is used in the WIP (see [8]).

An alternative to the HTTP server could be a WAP server. WAP, the wireless application protocol [23], is a protocol for transferring hypertext, especially designed for cellular phones. WAP is more lightweight than HTTP.

Muilet provider: Offering the MIDlets as files on an HTTP server also seems to be a good solution and the same approach could be used in a commercial system. The motivation as to why an HTTP server is a good choice is the same as for the Service provider.

Command manager: Implementing the Command manager with ASP scripts on an HTTP server was not a good idea. It was slow because an HTTP request was needed for each command and since HTTP is stateless a new socket has to be opened between the Palm handheld and the WIP for each command. We also wanted the ability to use *push* from the server, which means that the server sends information to the client on its own initiative when an event occurs. Examples of such events could be that the VCR comes to the end of the tape or that a new tape is inserted. With ASP scripts, the server can only send information back to the client as a response to an HTTP request. The response is included in the HTML output from the ASP script. Reacting to events on the server would require polling, which is not ideal.

As discussed in Section 5.5, a better solution would be to construct a specialised server daemon which follows a new protocol. The daemon opens a TCP/IP socket when a MIDlet wants to connect and keeps the socket open during the whole lifetime of the MIDlet. All commands and responses would then be sent over that socket. This would have solved both problems discussed above. The user only has to wait once for the relatively long time it takes to establish a TCP connection. Sending the data over the socket is fast after that. Since the socket is open the whole time, push is possible. The server

sends data over the socket when something happens. A specialised daemon could not be implemented in the WIP, so these ideas have to wait until later projects.

Physical equipment: The PLC worked as expected, but would not, as mentioned above, been needed in the optimal solution where the Bluetooth circuits had been inside the VCR.

In this optimal solution, the Server hardware component would have been a chip inside the VCR. In this chip, the Bluetooth stack and the other components in the Communication layer and Application layer would have been integrated. Large-scale production would make this chip quite cheap and manufacturers of home electronics could afford to put it in their products. This is part of BlueCell's vision.

The lack of two-way communication in our implementation, discussed in Section 5.5, is a serious drawback. This has to be solved in a commercial system. For the advantages of this kind of remote control to be exploited, it must be possible to show dynamic information from the server device on the client device. Two-way communication is of course necessary in order to use push from the server as discussed above.

6.2 Security

Security has not been an issue in this project, but is certainly very important in real applications. There are two aspects: The first aspect is that all users should not have the right to control all equipment (e.g., most users do not want the channels on their TV to be changed by their neighbours). Passwords or some other kind of verification might be needed. Bluetooth facilitates this to some degree with a mechanism called *pairing*, meaning that after establishing a first connection with authentication, two devices recognise each other as trusted.

The other aspect is that you want protection against malicious code. One approach here is that the user makes sure to only download MIDlets from trusted sources. This could be enforced by, e.g., digital signatures and certificates. Complementary security can be achieved by having protection in the JVM and execute the MIDlets in a "sandbox", as is used in MIDP. This is the same principle that applets use.

6.3 Usability

From the user's perspective, our prototype system would not be preferred over current VCR remote controls. An ordinary remote control works the

instant the user picks it up. Despite the effort to make our system easy to use, several actions have to be performed before the remote control starts to work. It takes time for the network connection to be established. To make the system attractive the startup has to be fast and simple.

Here, we can relate to the discussion about ubiquitous computing in Chapter 1. The ideal is that the remote control is easily available when the user wants it. At other times the user should not be disturbed by it. It would be a good thing if the discovery of server devices could run in the background, but then the notification to the user of a new found device must not be too strong.

Another aspect here is the integrity of the user, which relates to the discussion on security above. It is important that services are not invisibly downloaded to the client device. The user must have control over the process.

Chapter 7

Related Work

7.1 Research Project

This master thesis is a preliminary study to a research project, which will be carried out at the department of Computer Science at Lund Institute of Technology within Lucas [16]. The project is entitled Migrating User Interfaces, and it will look at how mobile devices can interact with each other in ad-hoc networks and present a combined user interface that enables use of the devices as a system, rather than as individual devices only. This should work not only with devices that are pre-configured to cooperate, but also with devices that were built independently.

The simplest case, looked at in this thesis, is controlling one server device from one client device. An example of a more complicated scenario is that the user sends a picture from a digital camera to a printer via a cellular phone, and can control everything from a user interface in the phone.

7.2 JayTalk

JayTalk is BlueCell's main product. It is a wireless platform that makes discovery of previously unknown services possible. If the user wants to, he can download the service to his cellular phone or handheld computer and use it. JayTalk is protected by a patent.

The problems we have looked at in this project are the same as those targeted by the JayTalk architecture. Our work is based on BlueCell's experience, and Mui has been strongly inspired by JayTalk. Still, Mui is not JayTalk—Mui does not follow the JayTalk specifications.

JayTalk has been designed to be independent of the wireless carrier. It works with, e.g., Bluetooth, WLAN and GPRS[1]. JayTalk is compliant with

[1] A WLAN is a wireless LAN, a wireless local area network. The technologies of wireless LANs are specified in the standard IEEE 802.11. GPRS, General Packet Radio Services,

J2ME, and uses CLDC and MIDP. MIDlets are transferred from servers to clients.

JayTalk is divided into three products, as listed below. Figure 7.1 shows where the different products are used.

JayTalk Navigator: This is the client part, an application that runs on the user's cellular phone or handheld computer. The user can discover, download and start services. MuiDiscovery in Mui corresponds to JayTalk Navigator.

JayTalk Server: JayTalk Server handles storage and distribution of applications to users. In Mui, it is represented by the Advertiser, Service provider, Muilet provider and Command manager components. A JayTalk Server can be built into a product, such as an audio system, or put on a desktop computer that acts as a server in a LAN. In the latter case, clients connect to the LAN via a LAN access point and then to the JayTalk server over the LAN.

JayTalk Administrator: This part makes it possible for a user to administrate JayTalk servers. It has a web interface and runs on a PC. JayTalk Administrator has no counterpart in Mui.

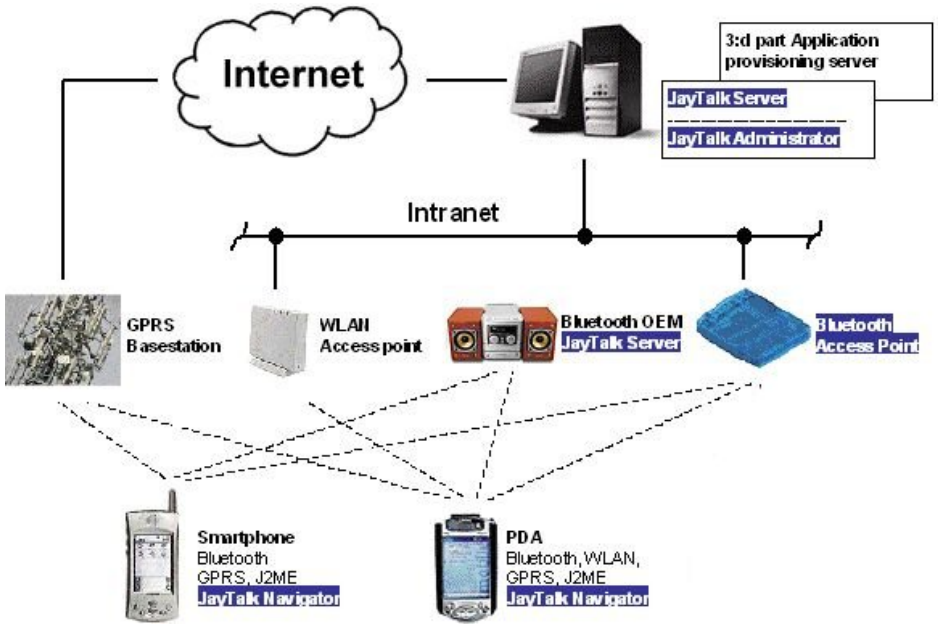

Figure 7.1: *JayTalk.*

is a packet based wireless communication service that gives higher data rates than GSM for cellular phones.

JayTalk has been implemented in an earlier project at BlueCell. A Compaq iPAQ, rather than a Palm handheld, was used as client device. That project involved porting the IVM virtual machine, discussed in Chapter 3, to Windows CE, the operating system used in the iPAQ. There has also been a master thesis project at BlueCell, where a JayTalk server was implemented [2].

The main reason for choosing to implement our own architecture, and not JayTalk exactly, was that we did not have access to server hardware that could be programmed in detail. BlueCell's existing JayTalk server could not be used either, because the versions of the Bluetooth protocols differed between the existing server and the Palm Bluetooth Card (Bluetooth 1.0B vs Bluetooth 1.1).

7.3 Jini

Jini is a technology developed by Sun that enables devices to form "spontaneous networks". The idea is that a device that is connected to the network should automatically be detected and detect all other devices on the network. New clients should then be able to use the services of all the other devices on the network. One example could be that a user connects his laptop computer to a new network. His laptop discovers a printer that the user can use to print out documents. What happens is that a printer driver written in Java is downloaded from the printer to the laptop. This ensures platform independence [20].

The limitation of Jini, as compared to the Mui project, is that you have to know in advance what kind of devices you want to control. In the above example it works well as long as it is a printer, but the user can not connect to a device that he is not prepared for, e.g. a digital camera.

7.4 UbiControl

UbiControl is a project at the Department of Computer Science at Swiss Federal Institute of Technology (ETH) Zürich [28]. The project is similar to Mui, in the sense that they are also trying to control consumer devices from a handheld computer. However, they have chosen a slightly different approach to the discovery problem. Instead of having the user choose from a possibly long list of services, the user simply points his handheld computer towards the device, and a laser pointer on the handheld computer is used to select which device to use. This is very intuitive as the user does not have to mentally connect the physical device to its virtual representation. The user will not be confused as three TV icons, possibly with the same name, pop up on his handheld computer. This reasoning comes from [27]. The disadvantage is that the user has to be aware that the

device exists and that it is controllable. The user cannot search the neighbourhood for controllable devices in the same simple way. There must be free line of sight to the device and the handheld computer has to be directed towards the device when selected. Perhaps a combination of this approach and ours would be good.

7.5 Wireless Microservers

Wireless Microservers is a joint project between Nokia and the University of Dortmund, discussed in article [10]. Wireless Microservers has many similarities with Mui. It is about storing a remote control user interface on a server, that is either built into a device or connected to a device, and controlling the device from a cellular phone, which first downloads the user interface over the air. The Wireless Microservers team have chosen Bluetooth for the wireless communication, and they use server-side scripting (like our ASP scripts) for controlling the device.

The main difference between the Wireless Microservers implementation and our solution is that the user interface is built as hypertext, WML pages, instead of implemented in Java. The server is a WAP server. The user brings up the remote control interface in a WAP browser on the phone and controls the device. The designers decided not to use Java because of its greater processing power demands. Although, they do see Java as an interesting option for the future.

An interesting idea in the Wireless Microservers project is that of a *pluggable server*. The designers argue that Bluetooth chips with integrated servers may, at least over the next few years, be too expensive to integrate them into low-cost devices. The pluggable server concept means that devices are manufactured not with a built-in server, but with a standard control interface. Then, a pluggable server can be bought if needed and connected to the device via a standard connector. The pluggable servers should be general-purpose, built for controlling any device. This enables cost-effective manufacturing in large quantities. The servers are general-purpose, but the user interface for a special device has to be distributed to the server at some point. The designers discuss different solutions for this: pre-programming a server with user interfaces for many devices, storing the user interface in ROM on the device and uploading it to the server at plug-in, or downloading the user interface over a network, either with the use of a PC or via a cellular phone and Bluetooth.

7.6 The Pebbles Project

The Pebbles project is carried out at the Carnegie Mellon University in Pittsburgh, USA [21]. It is about turning a handheld device, such as a handheld computer or a cellular phone, into a *personal universal controller*

(PUC). The PUC can control home, office or factory equipment. A specification of the appliance's functions is downloaded to the PUC, which creates a user interface, and there is two-way communication between the PUC and the appliance. The user interface is adapted to the properties of the handheld device (display type and input mechanisms) and to the user's preferences. The specification language is based on XML and the description of the user interface is at a high level, which gives flexibility.

The ideas in the Pebbles project are very similar to the ones in our project. A difference is that an abstract description is used for the interface, and not Java as in our project.

Chapter 8

Conclusions and Further Work

8.1 Conclusions

The technology envisioned for this thesis has turned out to be feasible. We have succeeded in constructing a functional system, where services in the proximity of a handheld device can be discovered, downloaded and started. In our scenario, a VCR can be controlled from a generic handheld device using the SonyVideo MIDlet, and the user interface adapts to different screen sizes and input capabilities. Furthermore, the technologies we have used in this prototype have worked well together.

Using Java to represent the user interface was a solution that worked well in our project. Java makes it easy to use the same user interface on a large number of devices, since JVMs are common.

The Bluetooth technology worked well in the system we built. Bluetooth is intended for ad-hoc networks, which this system relies on. We could benefit from the existing inquiry mechanisms when discovering Mui services.

Much effort was spent trying to make the user interface description flexible, so that the user interfaces could be displayed on any device. The result was the `muigui` library, which has two modes: the graphic mode is a graphical user interface, and the menu mode is a hierarchical menu system.

The solution we have implemented on the server side is a prototype built out of existing hardware components, and it is not the one we want to suggest. The functionality of the equipment that we have used outside the VCR should be realized inside it. Using an HTTP server for providing the list of services and the MIDlet files was good, but for controlling the VCR, the HTTP server was not optimal. For the latter, a server providing both push and pull is needed and probably a specially written daemon.

8.2 Further Work

Our work has been to put together a prototype for demonstrating the feasibility of the techniques in one particular situation. To make the techniques available in a commercial setting, improvements would have to be made in a number of areas.

Single chip: As mentioned above, all external equipment on the server side should be made smaller and placed on a single chip, possibly together with a Bluetooth implementation, which could be put inside the server device.

Two-way communication: To make the most of the advantages of a dynamic remote control, two-way communication is needed. Information from the server device should be possible to display on the screen of the handheld device.

Push: To enable the server device to notify the client device with little delay when an event occurs, a push protocol has to be implemented.

Performance: The performance of the system has to be improved. This applies both to the download time and to the response time when interacting with the equipment.

Usability: To make the system appealing to the user, it has to be as simple to use as an ordinary remote control, or simpler.

Security: The security issues are important and must be considered in future systems.

For a system like this to gain acceptance, a standard must be established, so that devices from different manufacturers can function together.

In Section 7.1, we discussed future work in this area at the department of Computer Science at Lund Institute of Technology. The plan is to expand the problem and look at more than two devices which can cooperate.

Appendix A

User Manuals

A.1 MuiDiscovery

MuiDiscovery is a Palm OS 4.1 application developed as part of this thesis work. The handheld has to be Bluetooth-equipped to work. The program is used to discover and start using Mui services offered by equipment within range of the handheld device.

Installation

To prepare the Palm handheld for Mui, the PRC files *MuiDiscovery.prc* and *xmllib.prc* should be installed on the Palm handheld, using HotSync.

Sun's MIDP for Palm OS Java runtime, with the KVM, is also needed. It can be downloaded from `http://java.sun.com/products/midp4palm/`.

Discovering Services

After installation, every time the user wants to use a nearby Mui service he just starts the MuiDiscovery application by tapping its icon in the Application Launcher. See Figure A.1.

Figure A.1: *MuiDiscovery's icon in the Application Launcher.*

MuiDiscovery starts and begins to search for Mui services. As it finds Mui server devices, it downloads lists of services from them. This will show as a status dialog popping up on the screen of the handheld. A password may have to be entered if a server device is password protected.

As lists of services are downloaded, MuiDiscovery displays them on the screen. See Figure A.2. This process takes about 30 seconds.

Figure A.2: *A list of services shown in MuiDiscovery.*

Sometimes, the sequence of events may not follow the description above. It might be that MuiDiscovery does not find any Mui server devices. Then, a message will be shown and the user can try again by choosing Discover Again on the Actions menu. This menu item can be used at any time to re-discover services. It can also happen that a Mui server device is found but that the connection or the download fails. This will result in another message, explaining the situation. In this case, the user tries again by choosing Download Again on the Actions menu.

Starting a Service

When lists of services have been downloaded, the user can start a service by tapping its icon in the list. The KVM will then run the service.

A.2 SonyVideo

SonyVideo is a MIDP 1.0 MIDlet suited for Palm OS. It is used to control a Sony SLV-SE220D VCR. The Palm handheld has to be Bluetooth equipped, since the MIDlet uses Bluetooth to communicate with the VCR.

Installation

The MIDlet is installed and launched with the program MuiDiscovery described in Section A.1. When the MIDlet has been installed on the Palm handheld, it can also be launched from the Application Launcher in the same way as normal Palm applications are launched, without the need to use MuiDiscovery.

Usage in Graphic Mode

When the program is launched in graphic mode it will look like Figure A.3. The user issues a command to the VCR by tapping a button with his stylus. The command is then sent to the VCR over Bluetooth. If a connection has not yet been established, there might be a delay while the handheld connects to the VCR.

Figure A.3: *The screen of SonyVideo in graphic mode. The user can tap his stylus on the buttons to issue commands to the VCR.*

Usage in Menu Mode

If the device does not have sufficient resources to launch the MIDlet in graphic mode, it will be launched in menu mode. This user interface might look very different on different devices, but the same general principle applies. The buttons found in Figure A.3 are organised in three screens,

sorted after which row they belong to. A main screen lets the user switch between screens. Figure A.4 shows the main screen and the screen for the topmost row.

Figure A.4: *The left figure shows the main screen of SonyVideo in menu mode. The user can navigate between the three different screens where he can select commands. The screen* ***System*** *is shown in the figure to the right. The user can choose between the two commands* ***On/Off*** *and* ***Eject****. He can also press* ***Back*** *to return to the main screen.*

Appendix B

`muigui` Overview

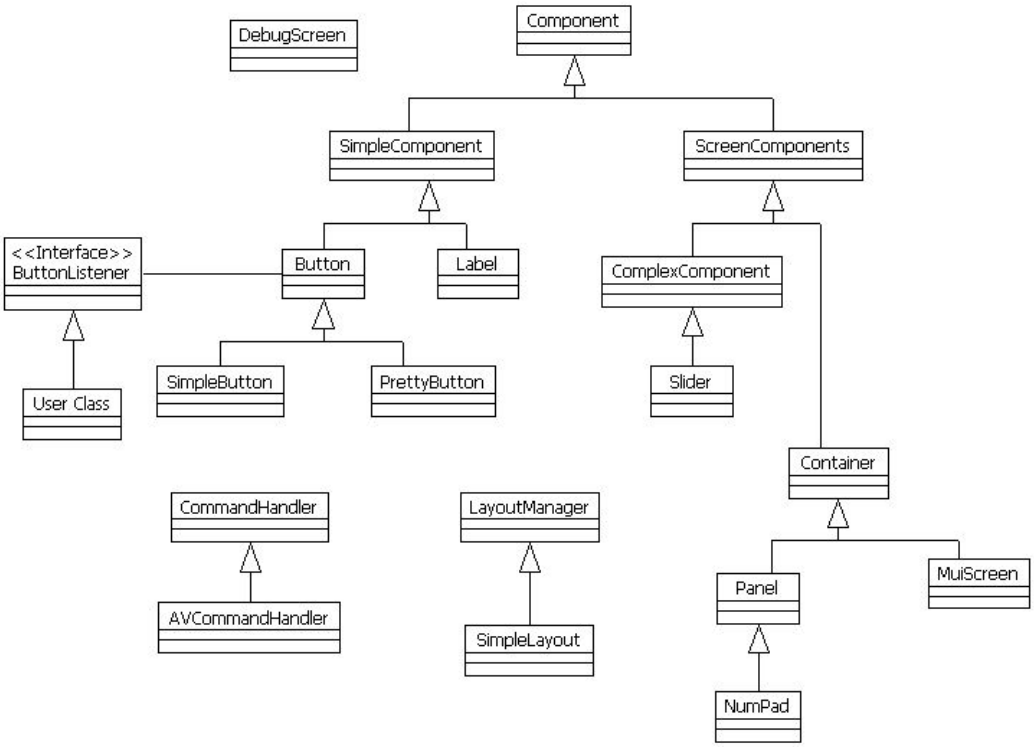

Figure B.1: *Class diagram of the `muigui` library*

Figure B.1 shows the class diagram of the `muigui` library.

Component Class that represents an object that can be placed on the screen.

SimpleComponent Class that represents a simple component. This is a component that does not need its own screen if the system is in menu mode.

ScreenComponent Class that represents components that have their own screen when in menu mode.

Button Abstract superclass to all different Button-classes.

SimpleButton Class that represents a very simple button.

PrettyButton Class that represents a Button that looks better than the more simplistic SimpleButton. There is a 3D effect and an image can be shown on the button.

Label Class that represents a text-string that can be placed on the screen.

ComplexComponent Class that represents an object that is more complex than SimpleComponents. A ComplexComponent can for instance be built up by several SimpleComponents.

Slider Class that represents a component where the user can select values by moving a handle.

Container Class that represents a component that can contain other components. The position of all components are decided by a LayoutManager.

Panel Class that represents a panel that can contain other components. The position of all components are decided by a LayoutManager.

NumPad Class that represents a collection of several buttons, which together form a numerical keypad.

MuiScreen The top-level component that contains all other components.

LayoutManager Abstract class that handles layout of a container.

SimpleLayout Class that describes a very simple LayoutManager. All components get an own row or column with some distance between them.

CommandHandler Abstract class that sends commands to the server when a button is pressed.

AVCommandHandler AVCommandHandler is a concrete implementation of CommandHandler. It communicates with the server according to the Bluetooth profile AVRCP.

ButtonListener Interface for classes interested in when buttons is pressed.

DebugScreen Class that represents a screen the programmer can write debug-information to.

Appendix C

Bluetooth

In this appendix, a short overview of Bluetooth is given. Bluetooth is a wireless technology for short-range, low-power radio communication, which has been used in this project. The information has been taken mostly from [6].

C.1 Background

Bluetooth was invented in 1994, at Ericsson Mobile Communications in Lund. It was named after the Danish king Harald Blåtand, who lived in the 10th century. Today, Bluetooth is a global standard, and its development is monitored by a Special Interest Group (SIG) that consists of nine promoter companies, including Ericsson, IBM, Intel, and Nokia, and more than 2000 adopter/associate member companies. Bluetooth capabilities are becoming more and more common in, e.g., cellular phones and handheld computers.

C.2 Profiles

Initially, Bluetooth was most used as a cable replacement, providing wireless serial communication between two devices. However, it is also intended for forming ad-hoc networks between devices, so that they can use each other's services, and for voice communication. These applications are becoming more common.

The different usage models of Bluetooth have been addressed by creating *profiles*. Each profile specifies how to use the Bluetooth protocol stack in a particular application. There are profiles for different kinds of serial communication, such as the Serial Port Profile, the File Transfer Profile, and the Object Push Profile, and profiles for networking, such as the LAN Access Profile and the Dial Up Networking Profile. There are also profiles for telephony.

C.3 The Technology

Bluetooth radio operates in a license-free band at 2.4 GHz. Data can be transmitted at about 720 kbits/sec, at a range of approximately 10 metres.

Both point-to-point and point-to-multipoint connections are supported. A Bluetooth device can connect to up to seven other devices at the same time. These devices form a *piconet*, where one device is master and the others are slaves. A *scatternet* can be formed by linking several piconets.

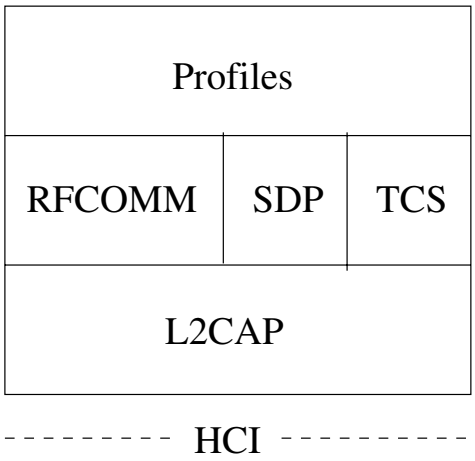

Figure C.1: *The software layers of the Bluetooth protocol stack.*

The software layers of the Bluetooth stack are shown schematically in Figure C.1. The Host Controller Interface (HCI) is an interface to the Bluetooth radio hardware, and the Logical Link Control and Adaptation Protocol (L2CAP) serves upper layers with, e.g., segmentation and reassembly of data packets. Above L2CAP, RFCOMM handles serial port connections, the Service Discovery Protocol (SDP) is for listing services offered by a node to other nodes, and the Telephony Control Protocol (TCS) is for telephony.

Appendix D

Dictionary

Ad-hoc Networks Networks that form spontaneously.

API Application Programming Interface.

ASP Active Server Page.

AWT Abstract Window Toolkit. It is a Java library that gives the programmer a platform-independent graphical user interface.

Bluetooth A wireless technology for short-range, low-power radio communication, which has been used in this project. Described further in Appendix C.

CDC Connected Device Configuration. A J2ME configuration for devices at the upper end of the spectrum, with at least a few megabytes of memory for the storage of the JVM and libraries.

CLDC Connected, Limited Device Configuration. A J2ME configuration intended for smaller devices than CDC with as little as 160 kB to 512 kB of available memory.

GUI Graphical User Interface.

GPRS General Packet Radio Services.

GSM Global System for Mobile communication.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

I^2C A de-facto standard for communication between integrated circuits in consumer electronics, invented by Philips Semiconductors [26].

IR Infra Red.

J2ME The Java 2 Platform, Micro Edition. Intended for small, resource-constrained devices.

J2SE Java 2 Platform, Standard Edition. Intended for standard desktop computers.

J2EE Java 2 Platform, Enterprise Edition. Intended for server applications.

JVM Java Virtual Machine.

kAWT An implementation of AWT for J2ME and KVM on Palm handhelds.

KVM Kilobyte Virtual Machine.

LAN Local Area Network.

MIDlet A MIDP application.

MIDP Mobile Information Device Profile. The only CLDC profile implemented so far. MIDP targets devices with at least 32 kB of memory available for the Java heap, a screen at least 96 pixels wide and 54 pixels high, and minimum input capabilities of a 0-9 keypad, arrow keys and a select-button.

Modbus A standardised protocol for controlling industrial equipment.

Palm OS The operating system used in all Palm handhelds.

Pervasive computing A synonym to ubiquitous computing.

PLC Programmable Logic Controller.

Ubiquitous computing A term meaning that computers are present everywhere in our daily environment. However, they should barely be noticeable for the user, she should be able to interact with them when she wants to without being disturbed at other times.

UIML User Interface Markup Language, UIML, is a language for describing a user interface that is to be shown on devices with different kinds of user interaction [37].

URL Uniform Resource Locator.

VCR Video Cassette Recorder.

WAP Wireless Application Protocol.

WIP Wireless Information Provider.

WLAN Wireless Local Area Network.

WML Wireless Markup Language.

XML Extensible Markup Language.

Bibliography

[1] Marc Abrams and Constantinos Phanouriou. UIML: An XML Language for Building Device-Independent User Interfaces. In *Proceedings of XML '99*, Dec 1999.

[2] Niclas Anderberg and Mikael Caleres. JTSmini—an embedded server for dynamically loaded wireless Java applications via Bluetooth. Master's thesis, Department of Computer Science, Lund Institute of Technology, 2001.

[3] Bluetooth SIG, Inc. *Audio/Video Remote Control Profile, v0.95b*, March 2002.

[4] Bluetooth Forum. `bluetooth-forum` at `news.palmos.com`, 2002.

[5] ConnectBlue AB. *Wireless Information Provider, User Manual 1.1*, 2002.

[6] Bluetooth Beginner's Guide. `http://www.ericsson.com/bluetooth/beginners_files/beginners_guide.pdf`, 2002.

[7] Everything2.org. `http://www.everything2.org`, 2002.

[8] GoAhead Software, the homepage of GoAhead Software, Inc. `http://www.goahead.com/`, 2002.

[9] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification*. Addison-Wesley, second edition, June 2000.

[10] Stephan Hartwig, Jan-Peter Strömann, and Peter Resch. Wireless Microservers. *IEEE Pervasive Computing*, 1(1), January–March 2002.

[11] Magnus Hjersing and Anders Ive. JAVAX, An Implementation of the Java Virtual Machine. Master's thesis, Department of Computer Science, Lund Institute of Technology, Dec 1996.

[12] Per Holm. *Objektorienterad programmering och Java*. Studentlitteratur AB, 1999.

[13] WebSphere Custom Environment. At IBM's homepage, `http://www-3.ibm.com/software/pervasive/products/wce/`, 2002.

[14] WebSphere Micro Environment. At IBM's homepage, `http://www-3.ibm.com/software/pervasive/products/wme/`, 2002.

[15] Magnus Landqvist. Porting and Evaluation of an Embedded Java Virtual Machine on Palm OS. Master's thesis, Department of Computer Science, Lund Institute of Technology, February 2000.

[16] LUCAS - Center for Applied Software Research. `http://www.lucas.lth.se`, 2002.

[17] Metrowerks Home. `http://www.metrowerks.com/`, 2002.

[18] The Modbus.org Website. `http://www.modbus.org/`, 2002.

[19] Motorola, Wireless Software, Applications & Services. *Java APIs for Bluetooth Wireless Technology (JSR-82), Specification Version 1.0a, Java 2 Platform, Micro Edition*, April 2002.

[20] Jan Newmarch. Overview of Jini. In *Jan Newmarch's Guide to JINI Technologies*, chapter 1. 2002. `http://www.javacommerce.com/tutorial/jini/Overview.html`.

[21] Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joe Hughes, Thomas K. Harris, Roni Rosenfeld, and Mathilde Pignol. *Generating Remote Control Interfaces for Complex Appliances*. Oct 2002.

[22] Nokia - Java Enabled Phones. At Nokia's homepage, `http://www.nokia.com/java/`, 2002.

[23] OMA, the Open Mobile Alliance Website. `http://www.openmobilealliance.org/`, 2002.

[24] Palm, Inc. *Palm OS Programmer's Companion*, July 2001.

[25] Palm.com, the homepage of Palm, Inc. `http://www.palm.com/`, 2002.

[26] Philips Semiconductors - I2C - I2C-bus. At Philips' homepage, `http://www.semiconductors.philips.com/buses/i2c/`, 2002.

[27] Matthias Ringwald. Spontaneous Interaction with Everyday Devices Using a PDA. Technical report, Distributed Systems Group, Department of Computer Science, Swiss Federal Institute of Technology (ETH) Zurich.

[28] Matthias Ringwald. UbiControl: Providing New and Easy Ways to Interact with Various Consumer Devices. Technical report, Distributed Systems Group, Department of Computer Science, Swiss Federal Institute of Technology (ETH) Zurich.

[29] Sony Ericsson licensierar Java från Sun Microsystems för multimediatelefoner. Pressrelease. At Sony Ericsson's homepage, `http://www.sonyericsson.com/se/`, March 2002.

[30] Sun Microsystems, Inc. *Connected, Limited Device Configuration, Specification Version 1.0, Java 2 Platform Micro Edition*, May 2000.

[31] Sun Microsystems, Inc. *Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices, White Paper*, May 2000.

[32] Sun Microsystems, Inc. *Mobile Information Device Profile (JSR-37), JCP Specification, Java 2 Platform, Micro Edition, 1.0*, September 2000.

[33] Sun Microsystems, Inc. *Over the Air User Initiated Provisioning Recommended Practice for the Mobile Information Device Profile, Version 1.0*, May 2001.

[34] Sun Microsystems, Inc. *Mobile Information Device Profile, v2.0 (JSR-118), JCP Public Draft Specification, Java 2 Platform, Micro Edition*, March 2002.

[35] Kim Topley. The Mobile Information Device Profile and MIDlets. In *J2ME in a Nutshell*, chapter 3. O'Reilly & Associates, Inc., March 2002.

[36] Hallvard Trætteberg. XMLlib — a small and simple XML parser. `http://www.idi.ntnu.no/~hal/development/palm/xmllib.html`, 2002.

[37] UIML.org. `http://www.uiml.org/`, 2002.

[38] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(30):94–104, 1991.

[39] Greg Winton. A Brief Tour of the Net Library. In *Palm OS Network Programming*, chapter 5. O'Reilly & Associates, Inc., September 2001.